Optoelectronic Fiber Webs for Imaging Applications

by

Jerimy Reeves Arnold

Bachelor of Science, Electrical Science and Engineering

Submitted to the

Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2005

Author . . . . . .
Department of Electrical Engineering and Computer Science
May 19, 2005

Certified by . . . . . .
Yoel Fink
Assistant Professor of Material Science and Engineering
Thesis Supervisor

Accepted by . . . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Optoelectronic Fiber Webs for Imaging Applications

by

Jerimy Reeves Arnold

Bachelor of Science, Electrical Science and Engineering

Submitted to the
Department of Electrical Engineering and Computer Science
on May 19, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

We demonstrate the use of novel visible and infrared light-sensitive optoelectronic fiber in the development of large scale photodetector arrays. Unlike conventional point photodetectors, these one-dimensional linear photodetectors are capable of sensing light along the entire length of the fiber and 360 radially. Multiple fibers can be arranged in an orthogonal grid to create a two-dimensional fiber web. The fiber web is capable of tracking a time-and space-varying beam, and output it onto a computer screen. Other imaging applications for the fiber web include image recovery for 2D images based on Computed Axial Tomography concepts, and lensless imaging. Lensless imaging is accomplished using two fiber webs separated by a fixed distance, recovering the intensity distribution on each fiber web, and applying a phase retrieval algorithm to the two distributions. Furthermore, fiber webs consisting of six planar arrays forming a cube can be used to detect incident light in three dimensions.

Thesis Supervisor: Yoel Fink
Title: Assistant Professor of Material Science and Engineering

Acknowledgments

I would like to thank my parents and family for their support through my academic endeavors.

I would like to extend my appreciation to Professor Fink for providing me with this research opportunity.

I would like to thank all members of the Photonic Bandgap Fibers & Devices Group, especially Ayman Abouraddy for his guidance and willingness to share his vast knowledge, Mehmet Bayindir for drawing the fibers used in this project, and Cathy Bruce for her administrative support.

Funding for this project was provided in part by Defense Advanced Research Project Agency (DARPA)/Carrano, DARPA/Griggs, DARPA QUIST, the ARO, the ONR, the AFOSR HEL-MURI, the US DOE, the ISN, and the Materials Research Science and Engineering Center (MRSCEC) program of the NSF.

Contents

1 Introduction **13**

2 Optoelectronic fibers **15**

2.1 Optical fibers as photodetectors 15

2.1.1 Optoelectronic fiber characteristics 15

2.2 Fiber webs: Two-dimensional photodetecting arrays 18

2.2.1 Advantages of a fiber web 18

2.2.2 Point detection with a fiber web 19

2.2.3 Vector detection with two fiber webs 20

2.2.4 Intensity distribution 20

3 Physical components of a fiber web **21**

3.1 Interfacing a fiber to an electrical circuit 21

3.1.1 Transimpedance amplifier: current-to-voltage converter 22

3.2 System considerations 23

3.2.1 Printed circuit board 24

3.2.2 Data digitization 24

3.2.3 Data transfer 25

3.2.4 Microcontroller 27

3.2.5 Analog multiplexer 28

3.2.6 -5 volt power supply 28

4 Software **31**

4.1 Microcontroller Code 31
4.1.1 USB descriptor 33
4.2 HIDCOMM plug-in 33
4.3 Application Software 33
4.3.1 Point detection 34
4.3.2 Vector detection 35
4.3.3 Intensity distribution 36
4.3.4 Imaging 37

5 Fiber web imaging applications **39**
5.1 Imaging with a fiber web 39
5.1.1 Recovering an image using CAT algorithms 40
5.1.2 Phase retrieval 41
5.2 Future work 42

6 Conclusion **45**

A Program code **47**
A.1 Microcontroller embedded program 47
A.1.1 Acquisition functions 47
A.1.2 USB Descriptor 53
A.2 3D Vector Detection 58

B Parts list **65**

C Full Circuit Schematic **67**

List of Figures

2-1 Cross-section of optoelectronic fiber taken with an SEM. The four electrodes can be seen in contact with the core, surrounded by the cladding with an outer diameter of 1 mm. 16

2-2 Incident photons generate electron-hole pairs in the core of the fiber. The induced voltage, V_o can be measured across the output resitor, R 16

2-3 (a) Electronic equivalent model of an optoelectronic fiber core. Each unit length of the core has an associated conductance, G. (b) Incident photon increases the conductance of section of the fiber core. (c) Multiple photons with equal intensity effect sections of the fiber core equally. (d) High intensity photon will have a larger impact on the conductance of the core. 17

3-1 Transimpedance amplifier with fiber input. 22

3-2 Photograph of the populated PCB. Outer dimensions are 12 inches square. 24

3-3 Circuit schematic of the PIC16C745 drawing power from the USB port. 27

3-4 The hardware design for the fiber web. 28

3-5 Inverting regulator which creates the negative supply voltage. 29

4-1 A screen shot of the fiber web functioning as a single point detector. . 34

4-2 A screen shot of two fiber webs each functioning as a point detector. The illuminated points on each detector are shown in blue and connected with a pink line. 35

4-3 Intensity distribution of an incident point showing how the intensity falls off radially from the center. 36
4-4 Screen shot of the imaging program. Each fiber voltage is displayed as a raw value (top) and normalized to the calibration value (bottom). . 37

5-1 Three illumination patterns all producing an ambiguous output. The power of the incident beams in (c) are half of those in (a) and (b). . . 40
5-2 Parallel projections of an arbitrary intensity distribution at different angles. 40
5-3 Normalized orthogonal parallel projections measured using a fiber web for an incident letter E. 41
5-4 Images recovered using data from a varied number of acquisition angles. 41
5-5 Transimpedance amplifier (a) with traditional feedback resistor and (b) a short section of fiber as the feedback resistor. 42

C-1 Schematic Page 1 68
C-2 Schematic Page 2 69
C-3 Schematic Page 3 70
C-4 Schematic Page 4 71

List of Tables

B.1 Electronics Parts 65

Chapter 1

Introduction

The purpose of this thesis is to develop two-and three-dimensional optical detector arrays from novel one-dimensional optoelectronic fibers that we shall call fiber webs. To fully grasp how such a detector can be advantageous, it is useful to describe the progression of how one gets from a single fiber detector to a multidimensional detector with multiple fiber webs. Once it us understood how a fiber web works, the imaging applications can be appreciated.

The design considerations made when building a fiber web are described in detail in chapter three. The considerations discussed include interfacing an optoelectronic fiber to an electrical circuit, data digitization, and data transmission.

Chapter four describes the design of the software used to control the fiber web, acquire data from the web, and display real-time analysis of the raw acquired data according to the desired application.

Chapter five describes imaging applications of the fiber webs and the algorithms required to interpret the data. The applications described are point detection, vector detection, and imaging using multiple intensity distributions. Possible future expansions to the project are also discussed.

Chapter 2

Optoelectronic fibers

2.1 Optical fibers as photodetectors

Conventionally a fiber is used as a conduit to transmit optical information through the fiber. The idea of using optical fibers as a photodetector arose from the development of a new type of fiber that is sensitive to visible and infra-red light along its entire length[1]. This new fiber has an optically sensitive core, four metal electrodes making intimate contact with the core running the length of the fiber, and a cladding surrounding the core and contacts (see Fig 2-1). Two of the four metal electrodes are contacted externally and connected with a series voltage source and resistor, allowing one to measure the impedance of the fiber, as shown in Fig 2-2. Upon illumination of the fiber, photons generate electron-hole pairs, and the conductivity of the fiber changes. This property will be exploited to develop the optoelectronic fibers into a photodetecting array.

2.1.1 Optoelectronic fiber characteristics

When used as a photodetector, the fiber is sensitive to light incident in the radial or axial direction. A fiber used as a photodetector will thus have a large area of detection as well as be able to detect light from 360 degrees radially. These optoelectronic fibers are produced by a thermal drawing process resulting in long continuous

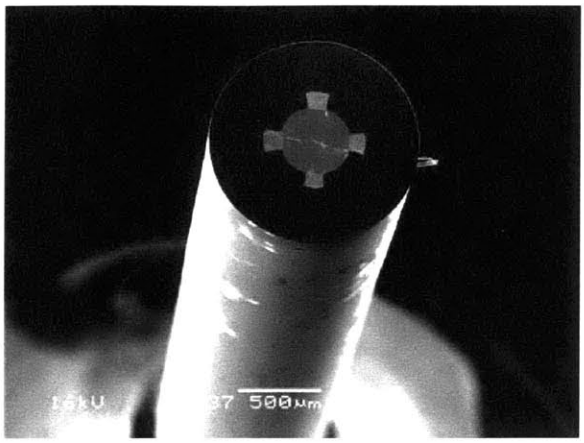


Figure 2-1: Cross-section of optoelectronic fiber taken with an SEM. The four electrodes can be seen in contact with the core, surrounded by the cladding with an outer diameter of 1 mm.

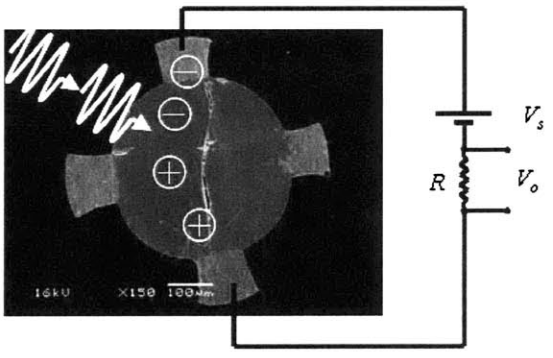


Figure 2-2: Incident photons generate electron-hole pairs in the core of the fiber. The induced voltage, V_o can be measured across the output resitor, R

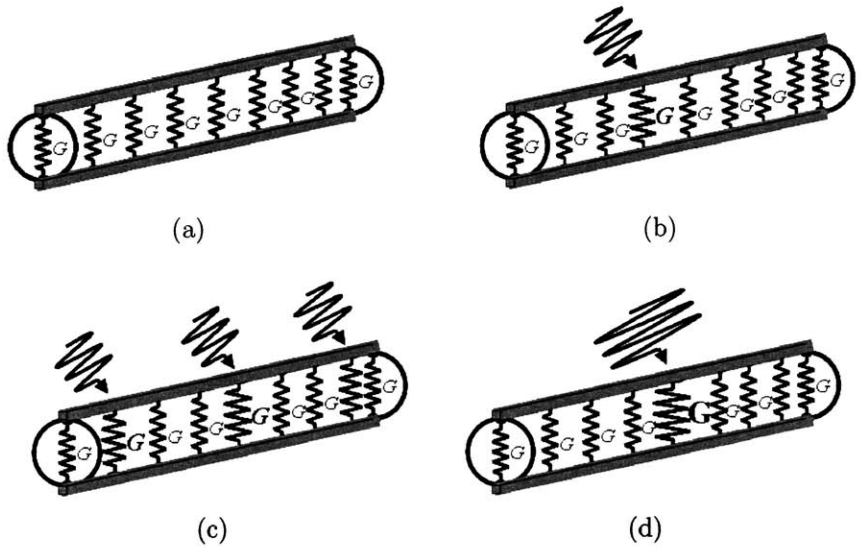


Figure 2-3: (a) Electronic equivalent model of an optoelectronic fiber core. Each unit length of the core has an associated conductance, G. (b) Incident photon increases the conductance of section of the fiber core. (c) Multiple photons with equal intensity effect sections of the fiber core equally. (d) High intensity photon will have a larger impact on the conductance of the core.

fibers, making them distinct from conventional single-point photodetectors. Another important property of the fiber detectors is that they are flexible and can be used to form detectors of arbitrary geometry.

The optoelectronic fiber acts as a line detector or integrator of power along its length. This can be seen more clearly if the fiber is modeled as many small conductances in parallel, each having a value G, as shown in Fig 2-3(a). Illumination of one conductance will increase the overall conductance of the fiber only slightly (see Fig. 2-3(b)). Illumination of many conductances will have a much greater impact on the overall conductance, as shown in Fig. 2-3(c). Fig. 2-3(d) shows that the change in conductance depends not only on the amount of illumination of the core, but also depends on the intensity of the illumination. For example, if a small area of the fiber is illuminated with a small amount of power, there will be a small change in the conductance of the fiber. A large area of illumination with large power will correspond to a large change in conductance. Remember that the fiber is an integrator, or a summation of conductances, and there are many possible combinations of illumination intensity and area that can result in the same fiber output. This relationship is

of extreme importance when discussing how to image with the fiber web.

From the fiber model the following equation relates conductance to the impedance of the fiber,

$$R_f = \frac{1}{\sum_{n=1}^{N} G_n}, \tag{2.1}$$

which is inversely proportional to the effective number of conductances, N, in the fiber, and N is directly proportional to the length. Therefore, a short piece of fiber with the same transverse dimensions will have a much larger impedance than a longer piece. This property has not yet been utilized, but will be discussed in the future work section.

2.2 Fiber webs: Two-dimensional photodetecting arrays

A fiber web is created by overlapping independent sections of fiber in an NxN square pattern, resulting in N^2 effective point detectors. One implementation of such a fiber web that we have produced is a 32x32 array with spacing between fibers of 7.5 mm. This results in a detection area of approximately 600 cm^2.

When used in a fiber web, the thin optoelectronic fibers are spaced a fixed distance apart creating a detector that allows the incident light to propagate beyond the detector with minimal disruption. This property of the photodetector gives fiber webs the unique ability to acquire incident intensity distributions at variable points along along a vector. A charge-coupled device, or CCD, is a conventional photodetector used to capture an intensity distribution incident upon the detector. Unlike a fiber web, a CCD absorbs the incident light, eliminating the possibility to acquire another intensity distribution at a second location.

2.2.1 Advantages of a fiber web

As stated before, a fiber web requires only 2N fibers to implement N^2 point detectors. Semiconductor detectors require the full N^2 number of detectors to get the same

resolution. The required number of fibers grows linearly with the desired number of point detectors. As the number of point detectors gets large, a fiber web has less required number of devices by almost an order of magnitude.

Although there has been some success in putting light emitting devices on curved surfaces by self-assembly[2], conceivably they could also be photodetectors. The flexibility of the optoelectronic fibers also allows for creation of curved surface detectors such as circular or cylindric detectors.

These properties give it multiple advantages over its semiconductor counterparts: detection on non-planar surfaces, detection over a large area, and reduction in the required number of devices. Conventional detectors are very small scale and the incident light must be focused with a lens. A fiber web allows images to be recovered without the use of a lens. Additionally, phase information can be recovered when imaging with a fiber web because of the ability to acquire data at multiple points along a vector.

2.2.2 Point detection with a fiber web

To use the fiber web as a detector, the detector must first be calibrated to the ambient light conditions under which it will be used. This calibration value will be used to normalize subsequent data to eliminate fluctuations between fibers due to variations created in the fiber drawing process.

When the fiber web is illuminated by a white light beam, the optical response of the fibers is measured and normalized to the value measured at ambient illumination. Each normalized deviation is compared against a threshold to determine if the fiber has deviated sufficiently from the calibration value to be deemed under illumination. Due to the nature of the fiber web, both a horizontal and vertical fiber must exceed the threshold for a point to be considered illuminated. In the event that multiple parallel fibers exceed the threshold, the fiber with the highest normalized deviation is selected as the illuminated fiber. Identifying a single point illustrates the capability of a fiber web as a large scale photodetector.

2.2.3 Vector detection with two fiber webs

A fiber web as a single point detector can be extended to multiple dimensions by adding additional fiber webs. Two fiber webs acting as point detectors spaced at a known distance can be used to identify a single point on each web. When a point is deemed illuminated on both webs, the points can be connected by a vector and the incident angle and direction of the beam recovered. With six fiber webs forming a cube, the same concept can be extended, encompassing the entire three-dimensional space.

2.2.4 Intensity distribution

As mentioned before, the spacing of the adjacent fibers in a fiber web is approximately 7.5 mm. A light source such as a flashlight can not focus the output beam onto a single fiber intersection. Although this issue does not create a problem when using the fiber web as a point detector, it realizes the fact that there is much more information in the fiber web that is not being displayed in the point detection configuration. The intensity distribution of the entire web is displayed as a color gradient, scaled by the normalized fiber deviation.

The intensity of each point is determined by the product of the normalized fiber voltages. The two orthogonal sets of normalized fiber voltages are each contained in a vector array. The product of one vector and the transpose of the other vector results in a matrix containing an intensity for all points of the array. This data is useful for single point intensity distributions. For example, displaying the resulting intensity of illumination by a flashlight. When there is more than one feature of an incident intensity, the resulting distribution on the fiber web is no longer a faithful representation of the incident intensity distribution. This results from the integration property of the fibers and we will rely on Computed Axial Tomography concepts to solve this issue.

Chapter 3

Physical components of a fiber web

Now that the concept of a fiber web is well established, the focus will be on the physical implementation. This involves interfacing the fibers to electronic circuits, digitizing the acquired analog voltages, and then transferring the acquired information to a computer for data processing. Due to the flexibility of the fibers, a physical structure to support the fibers is also required.

3.1 Interfacing a fiber to an electrical circuit

The impedance of a single fiber is on the order of 10 to 200 MΩ (depending on length and core diameter), making it impossible to connect directly to an analog-to-digital converter (ADC), generally a low input impedance circuit. Connecting a fiber directly to the ADC would load down the fiber by effectively placing a low impedance in parallel with the fiber impedance. The resulting impedance seen is effectively the input impedance of the ADC. Even though the same current exists in the fiber, the voltage would now be to low to measure, because the voltage would be on the order of the least significant bit of the ADC. In order to accurately measure the voltage on the fiber, a buffer circuit must be introduced that does not exhibit the same loading effect.

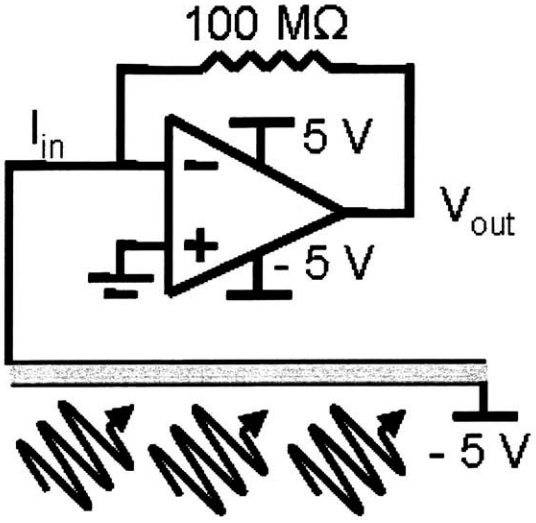


Figure 3-1: Transimpedance amplifier with fiber input.

3.1.1 Transimpedance amplifier: current-to-voltage converter

The circuit shown in Fig 3-1 uses an op-amp configured as a current to voltage converter, or transimpedance amplifier. The current that flows from the fiber must flow through the 100 MΩ feedback resistor connecting V_{out} to I_{in}, because of the high input impedance of the op-amp. Therefore, the output voltage is the product of the fiber current and the 100 MΩ resistance. The transimpedance amplifier is capable of accommodating a wide range of input currents because the feedback resistance can be varied to set the desired gain and DC output voltage of the circuit. For a fiber input and a feedback resistance of 100 MΩ, the DC output voltage ranged around 2.0 volts and the output varied over 1.4 volts for ambient versus heavy illumination.

Knowing that the current out of the fiber is always positive, the output of the amplifier will also always be positive, ranging from 0 volts to the supply. This is required to meet the input range of most analog-to-digital converters. In order to properly bias the amplifier while utilizing the full output of the amplifier, a +/- 5 volt supply is needed.

A major concern when interfacing the fibers to an electronic device is being able to detect the very low levels of *dark current* that exist in the fiber. Dark current is defined as the amount of current that will flow in the device under no illumination.

At a bias of 5 volts, measurements show an optoelectronic fiber has a dark current on the order of 10 nA. The input bias current of the op-amp must be much lower than this in order to accurately measure the fiber current[3]. The AD8625 from Analog Devices is a quad-package precision low power JFET input amplifier with a maximum input bias current of 1 pA. This amplifier will also operate on a dual 5 volt supply.

The feedback resistor connecting V_{out} and I_{in} will set the gain of the circuit and also the DC output voltage while the circuit is operating under ambient conditions. Since there is a DC voltage present at the output, it will not be possible to utilize the full 0 to 5 volt input range of the ADC. However, we want to maximize the output swing of the amplifier without saturating under any illumination conditions. The amplifier sensitivity, output swing, and DC output voltage will all increase as the feedback resistor is increased. Therefore, a balance between output swing and DC voltage must be set. This balance occurs when the feedback resistor is sized so the output is at approximately half of the supply voltage. Tests of the fiber show that the current will increase from 20 nA in ambient light to over 40 nA under heavy illumination. There is some slight variation amongst the fibers, so to be conservative, a maximum current of 50 nA is assumed. This results in a feedback resistance of 100 MΩ.

3.2 System considerations

An essential requirement of the fiber web is that it must be scalable; it can be used as a single device, or multiple fiber webs can be used simultaneously for two or three-dimensional vector detection or phase-retrieval. For example, universal serial bus (USB) connections are extremely scalable, allowing multiple devices to simultaneously transmit data to a host computer. Portability is also a concern for the fiber web, so external devices must be at an absolute minimum. This section discusses the system design considerations such as physical structure and data digitization while trying to maintain scalability and portability.

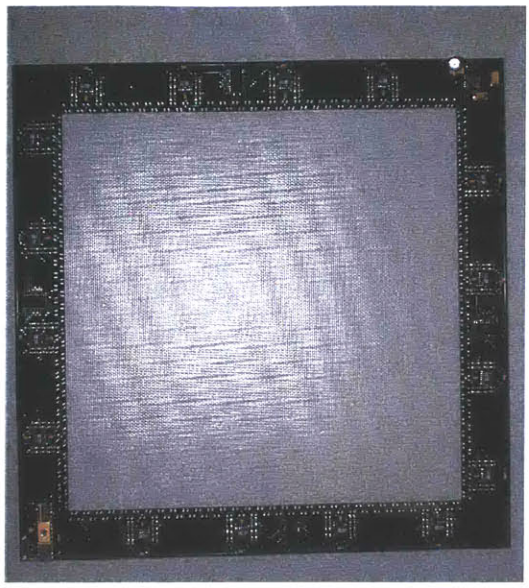


Figure 3-2: Photograph of the populated PCB. Outer dimensions are 12 inches square.

3.2.1 Printed circuit board

One of the requirements of the design is a support structure for the fibers. A printed circuit board would provide this function, as well as its major function of routing signals and holding electrical components such as the op-amps used to interface to the fibers.

3.2.2 Data digitization

The focus of the design now shifts to the digitization of the fiber voltages. The ADC options considered were a National Instruments data acquisition (DAQ) device or an integrated circuit (IC) version. The main advantage of using a DAQ device is the software that comes with the device allows for simple drag-and-drop programable applications for controlling and displaying information. Therefore, each fiber amplifier could be connected to an individual input of the DAQ device and the system would be complete. The drawbacks of such a device are high cost, not easily scalable to multiple devices, and a physically large device.

An IC ADC has very small physical dimensions and low cost, but software would have to be developed to display the data. An IC could also easily be placed on

the printed circuit board requiring no external wires, as is the case when using a DAQ device. A microcontroller would also be necessary to transfer the acquired data via serial or USB port. However, many microcontrollers now offer integrated ADC functionality, eliminating the need to interface the two separate devices.

The minimal detectible current from a fiber can be seen in the equation,

$$I_{min} = \frac{V_{max}}{R_f 2^N}, \tag{3.1}$$

and depends on the maximum allowable amplifier output voltage, the number of bits of precision in the ADC, N, and the value of the feedback resistor. For an 8-bit ADC, V_{max} of 5 volts and R_f of 100 MΩ, the minimum detectible fiber current is 200 pA. This level of accuracy will be more than sufficient for our application restricting our search of converters to require no more than 8-bits.

3.2.3 Data transfer

There are many protocols available to transmit data from an electrical device to a computer. As computers get smaller and the demand for higher data transmission rates continually increases, computer manufacturers are eliminating the older and slower communication ports such as serial and parallel ports and adopting USB as the single communication standard. USB has become the dominant communication for wired devices, but wireless communication popularity has also increased in recent years and may be a viable alternative.

Serial communication

Serial communication ports are capable of transmitting data at a maximum of 115 kilo-bits per second (Kb/s). Most microcontrollers have a port that can be configured to interface with a RS-232 driver/reciever circuit. However, serial ports are becoming less common on new computers, especially on laptops. Therefore, using serial communication would not be viable option for the fiber web because of compatibility issues.

USB

The USB 1.1 specification allows for data rates of 1.5 mega-bits per second (Mb/s) for low speed and up to 12 Mb/s for a full speed device[4]. The 2.0 specification adds high speed devices capable of transferring data at a maximum of 480 Mb/s[5]. These data rates are the maximum possible and the actual data rate will depend on the transfer mode used. USB supports four transfer modes: control, interrupt, bulk, and isochronous. Control transfers are generally used to relay status information and enumeration of a device on the bus. Interrupt transfer mode is non-periodic and is reliant upon the host polling the device before it can send data. A specific type of interrupt transfer mode is the human-interface device mode. The HID transfer mode is a low speed transfer generally used by devices such as mice or keyboards which do not have to transfer data at high rates. Bulk transfers, supported by high and full speed devices, are used to send large amounts of data at infrequent intervals. The highest data rates are achieved using isochronous transfer mode, also supported by only full and high speed devices. Data is transferred at a guaranteed bandwidth, periodically sending a constant amount of data.

An added benefit arises when using a USB connection for data transmission; the remote device is able to draw power from the host computer. A 5 volt supply at a maximum current draw of 500 mA is available to all devices connected to the port. Using a USB port will increase the scalability of the fiber web by not requiring external power supplies for each web and allowing for multiple webs to be connected to the same computer.

Wireless communication

Wireless communication has become an increasingly popular and reliable medium for data transfer. Bluetooth is a wireless communication protocol that operates in the unlicensed Industrial, Scientific, and Medical (ISM) frequency band of 2.4 to 2.483 GHz with a theoretical maximum RFCOMM data rate of 704 Kbps for the 1.1 specification[6]. The LMX9820A from National Semiconductor is a fully integrated

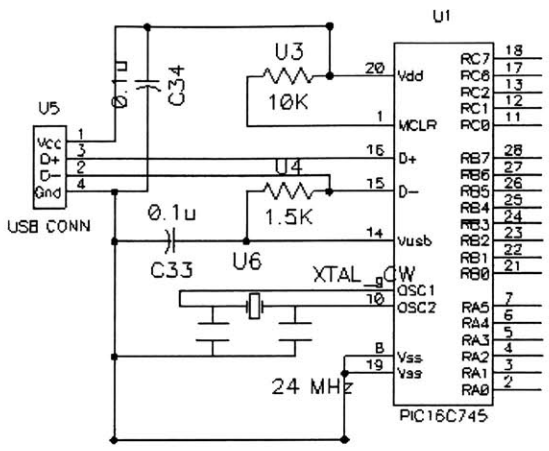


Figure 3-3: Circuit schematic of the PIC16C745 drawing power from the USB port.

baseband controller and radio providing a complete single chip solution. This device has a UART communications port enabling communication with a microcontroller, for example. This approach adds complexity and the first iteration of the fiber web will use a physical connection for data transfer. Wireless communication will be explored in future work.

3.2.4 Microcontroller

The Microchip PIC16C745 microcontroller is a fully programmable RISC microprocessor with built in USB communications port and integrated 8-bit ADC. This microcontroller has an internal clock requirement of running at 24 MHz in order to properly run the USB module. The 24 MHz clock can be set by either an external 24 MHz crystal, or an external 6 MHz source that gets multiplied up to the necessary frequency by an internal 4 times phase-locked loop multiplication. The external frequency mode must be specified in the microcontroller program code for the processor to function correctly.

The PIC16C745 has 22 input/output channels; five of these channels can be used as inputs to the ADC. A schematic of the PIC16C745 with all required external components is shown in Fig. 3-3. In order to maintain 8-bit accuracy with the ADC while running the USB module at 24 MHz, the maximum sample rate is limited to

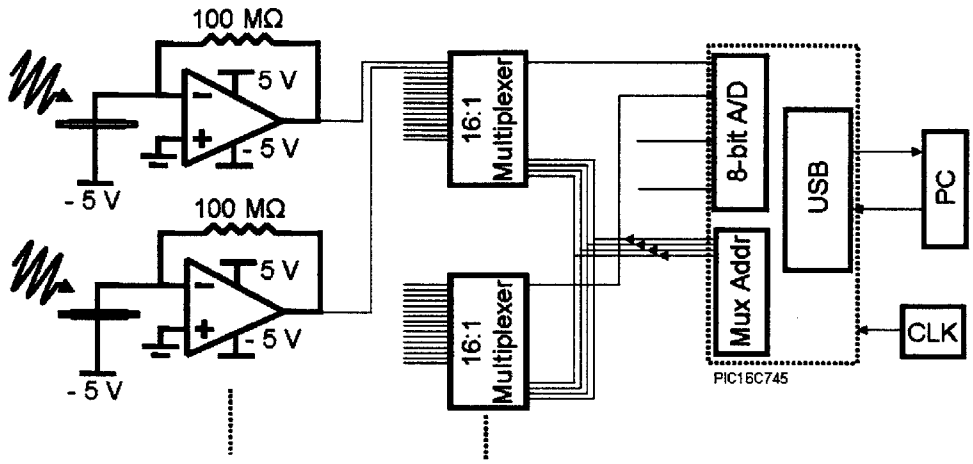


Figure 3-4: The hardware design for the fiber web.

8.3 kilo-samples per second (ksps).

Microchip supplies a programming environment to develop processor code for the microcontroller. The program can be written in assembly or C. However, a C compiler is not included with the programming environment and must be purchased from an external source.

3.2.5 Analog multiplexer

The 64 fiber inputs must be multiplexed down to 5 or fewer outputs so that the microcontroller can sample all of the fiber channels. This should be done in the fewest number of layers possible to avoid long cascaded switching delays. The CD74HC4067 is a 16 to 1 analog multiplexer/demultiplexer that will allow the 64 fiber inputs to be sampled by four channels of the microcontroller ADC using four of these multiplexers. The microcontroller supplies one of sixteen 4-bit addresses; each address controls a single input that will be sent to the multiplexer output. The overall system architecture can be seen in Fig. 3-4.

3.2.6 -5 volt power supply

The transimpedance amplifier used interface the fibers to the ADC requires a +/- 5 volt supply. The positive supply is obtained directly from the USB port as per the 1.1

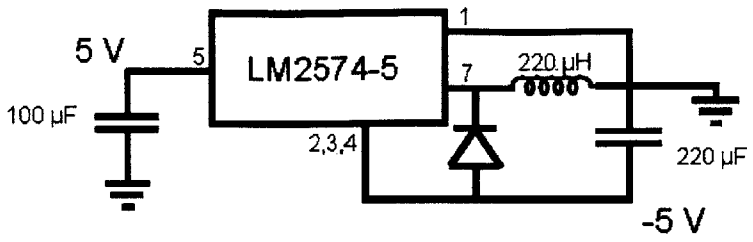


Figure 3-5: Inverting regulator which creates the negative supply voltage.

specification. The negative supply is not readily available, but can be created with a power converter. A buck-boost, or inverting regulator, creates a negative supply voltage from a positive voltage; in this case -5 volts from a positive 5 volts. Fig. 3-5 shows the topology for an inverting regulator using the LM2574-5 switching regulator from National Semiconductor.

Chapter 4

Software

The software required for the fiber web to reliably communicate data to a host computer involves three layers. The first layer of software is compiled and burned onto the microcontroller. This layer includes the required USB descriptors and protocols for compliance with the USB specification. The second layer is the HIDCOMM Active X plug-in that communicates between the USB host driver and the third layer, the user application software. Multiple software programs were developed in Visual Basic to accommodate various fiber web applications: point detection, intensity distribution plotting, vector imaging, and imaging.

4.1 Microcontroller Code

The microcontroller code is written in the C programming language and can be seen in the appendix. The preprocessor code includes setting the input/output port directions, defining the bits corresponding to the 16 multiplexer addresses, defining the USB maximum send and receive packet size, and including the proper USB descriptor and header files. The USB descriptor and header files were obtained from USB example code included with the CSS compiler package. We will be confined to using a maximum packet size of 64 bits, or 8 bytes. This packet size definition must correspond with the definition in the USB descriptor file for the packets to be sent correctly. Eight bytes is the maximum packet size for low speed USB devices. We

are limited to being a low speed HID device because of the complexity of writing a custom Windows driver. Windows has a standard HID driver that can be used with any HID device such as mice and keyboards. The drawback is we are limited to the 8 byte packet size at a maxim polling rate of 10 ms as per the USB specification, or 6.4 kbps.

The microcontroller program will first try to enumerate the device with the USB host. If it is unable to enumerate, the program will halt. After the device is enumerated, it waits for a signal from the user program to run a short loop acquiring data from the fibers. This loop acquires 20 samples from each fiber to be used as the calibration data. This process is facilitated by three functions: *sample_data, muxsel, and adc_sel. Muxsel* is a function that takes an integer argument from 0 to 15 and sets the multiplexer address according to the argument. *Adc_sel* is structured the same as *muxsel*, but configures one of the four ADC ports for data acquisition. *Sample_data* takes an integer argument equal to the number of fibers in the array, in this case 64. *Sample_data* calls on *muxsel* and *adc_sel* to acquire the data sequentially and store it in one of eight buffers. The acquisition order of the data does not correspond directly to the fiber order, so an algorithm was devised to place the data in the proper buffer location. After all the buffers have been filled, they are sent to the USB port using the vendor defined function, *usb_put_packet.*

Once calibration is complete, the microcontroller waits for two packets from the host computer to identify the data acquisition mode. There are two basic modes of operation with only a small deviation between the two: a continuous loop for point and vector detection and a short burst mode for imaging. The continuous loop allows real-time display of the acquired data for the desired output application. The burst mode was added to allow for rotation of the projected image to the desired angle between samples. Both of these modes use the functions described in the calibration routine.

4.1.1 USB descriptor

The USB descriptor contains all of the pertinent information required to identify the device to the host controller. This includes the maximum packet size for each endpoint, vendor identification number, product identification number, serial number, etc. In order to connect multiple fiber webs to one computer, the HIDCOMM plug-in requires that each device have unique identifiers in each category. A unique descriptor is therefore created, compiled, and burned into each fiber web microcontroller.

4.2 HIDCOMM plug-in

The HIDCOMM Active X plug-in is the bridge between the USB host and Visual Basic that allows communication with the microcontroller on the USB port without a custom software driver. The plug-in is distributed by Microchip for use in custom applications such as ours. Multiple instances of the plug-in can be used in a single application to acquire data from multiple devices. To link a device with a HIDCOMM instance, the device must be plugged in and recognized by the computer. (A properly programmed device will be recognized by Windows and configured automatically.) Once the device is configured, right clicking the HIDCOMM plug-in will display a property window containing all of the configured USB devices. Selecting the desired device will display all of the information specified in the descriptor and link the device to the plug-in.

4.3 Application Software

Several user interfaces have been developed, each for a specific application of the fiber web. The HIDCOMM plug-in provides send and receive functions to communicate with the microcontroller storing the received data into user defined buffers. The acquisition and storage of the fiber data is identical for all of the fiber web applications, the interfaces differ in how the data is interpreted and displayed. Since the program code is almost identical, only one instance is included in the appendix.

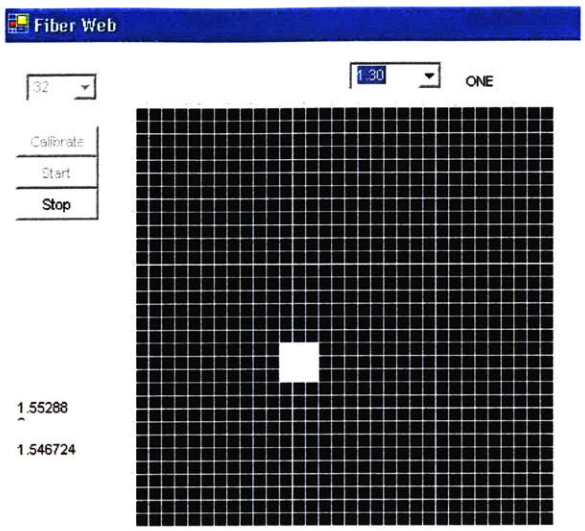


Figure 4-1: A screen shot of the fiber web functioning as a single point detector.

4.3.1 Point detection

The fiber web applications grew in a successive progression each building on the previous achievements, adding complexity as the task at hand increased in difficulty. Point detection is the first application in this progression and is designed to track the location of a time varying incident beam as it illuminates a point of the fiber web.

All of the images for the user interfaces are created with the Windows graphics libraries in Visual Basic .Net. The background for the point detection interface is a 32x32 black square pattern separated by white lines. Each square represents a pixel, or the intersection of two orthogonal fibers on the fiber web. Three buttons are located on the left side of the pixel array; *calibrate*, *start*, and *stop*. The *calibrate* button initiates the acquisition of 20 samples for each fiber. This data is sent from the microcontroller to the array program where it is averaged and stored in the calibration array. This calibration data is used to normalize all subsequent data samples. After the calibration routine has completed, the *start* button becomes active. This button will start a continuous loop running until the *stop* button is depressed. Each cycle of the loop acquires a new set of data for each fiber which is normalized by the stored calibration data. The loop steps through the data checking if the normalized value of the fiber exceeds a specified threshold. If the normalized value surpasses the threshold,

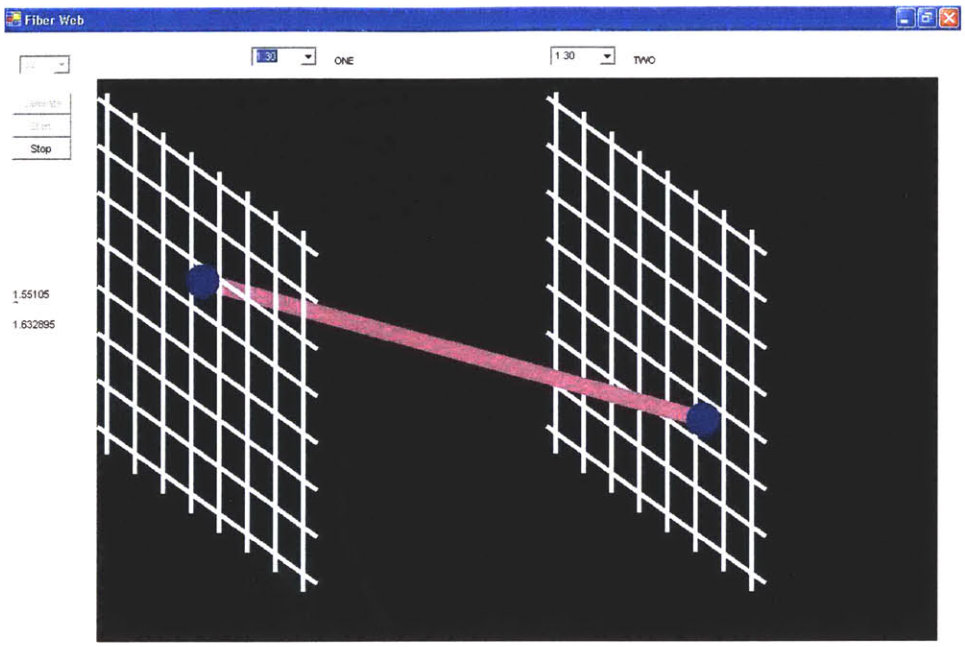


Figure 4-2: A screen shot of two fiber webs each functioning as a point detector. The illuminated points on each detector are shown in blue and connected with a pink line.

the fiber location is stored and the normalized value becomes the new threshold. This process is done separately for the rows and columns. If none of the normalized values surpass the threshold, no point is deemed illuminated. If multiple parallel fibers exceed the threshold, the algorithm guarantees the fiber with the highest deviation is the illuminated fiber. The location of the illuminated point is identified as a white 3 by 3 square on the detector display. The threshold is reset at the beginning of each loop and the initial value can be set a priori or during run-time using the threshold combo box on the user interface.

4.3.2 Vector detection

Vector detection uses two fiber webs that are each functioning as individual point detectors. The point detector software is extended to accommodate both fiber webs by adding an additional HIDCOMM plug-in configured for the second web and extending the existing point detector algorithm to handle the additional data. The user interface is modified to portray the webs in a three-dimensional space (4-2). When a point is detected on the front (x_1, y_1) and rear webs (x_2, y_2), a line is shown on the

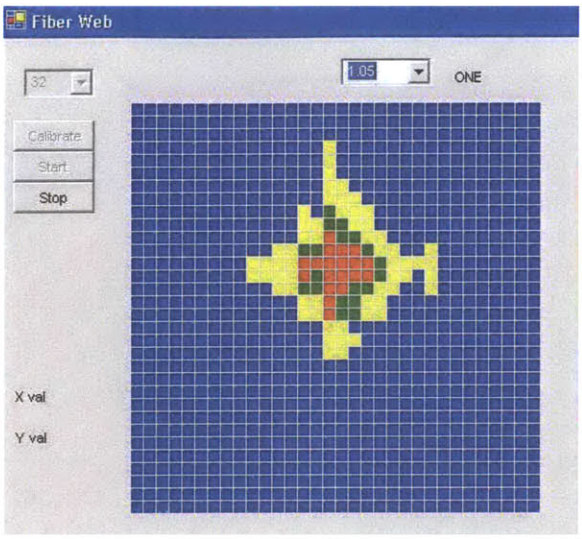


Figure 4-3: Intensity distribution of an incident point showing how the intensity falls off radially from the center.

user interface connecting the two points. When a known distance, d, separates the two webs, the horizontal (Θ_x) and vertical (Θ_y) angles can be determined from the equations,

$$\Theta_x = \arctan \frac{x_2 - x_1}{d} \tag{4.1}$$

$$\Theta_y = \arctan \frac{y_2 - y_1}{d} \tag{4.2}$$

.

4.3.3 Intensity distribution

When a single illumination point is not sufficient, an intensity distribution can be plotted giving information about the entire array. A value for each point $p_{(i,j)}$ can be computed from the normalized row and column data from the equation,

$$p_{(i,j)} = (1 - r_i)(1 - c_j) \tag{4.3}$$

where r_i is the normalized fiber voltage for the i^{th} row and c_j is the j^{th} column fiber.

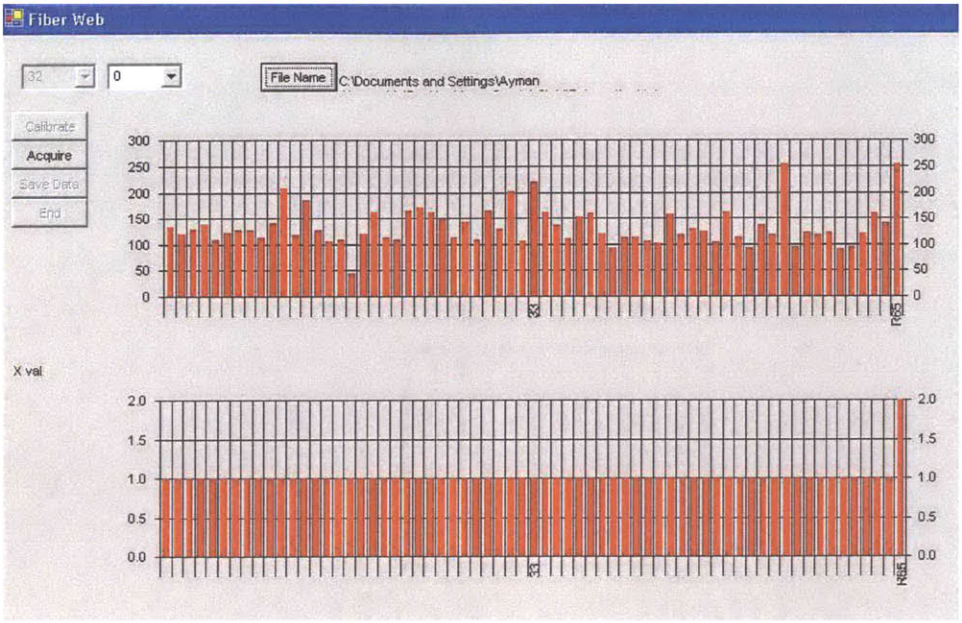


Figure 4-4: Screen shot of the imaging program. Each fiber voltage is displayed as a raw value (top) and normalized to the calibration value (bottom).

The user interface is identical to that of the point detector except a few minor changes. The threshold combo box contains values that are representative of those seen for individual point intensities. The second change cannot be seen until the web is illuminated. The individual squares change color from their default black, to a color corresponding to the intensity ranging from blue to red, with red being the highest intensity as shown in Fig. 4-3.

4.3.4 Imaging

Imaging with the fiber web is detailed in the following chapter but the software can be discussed without knowledge of the imaging process. The imaging program has a unique feature of saving the data to file. The image recovery algorithm is math intensive and the data is processed in Matlab. Specifying a file is required before measurements can be taken. Like all of the other programs, the imaging program has a *calibration* button to acquire fiber data under ambient illumination conditions. Unlike the other programs, the imaging program does not run a continuous loop to acquire data, but uses short bursts. These bursts are initiated by the *acquire* button

and acquire 20 samples from each fiber which are averaged in the imaging program. Fig. 4-4 shows the imaging user interface which has two sets of bar-graphs that display the individual fiber data. The top graph is the averaged data and the bottom is normalized to the calibration data. After each acquisition, the *save data* button can be pressed to save the data to a specified file. Because rotation is required in the image recovery process, a combo box is provided to record the angle for each data acquisition. The *end* button closes the communication with the microcontroller and outputs all of the saved data to the specified file.

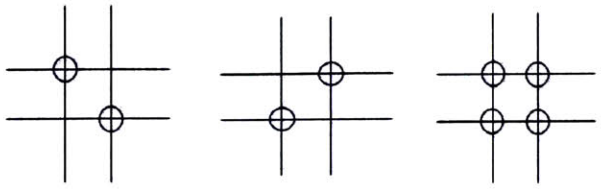
Chapter 5

Fiber web imaging applications

Unique imaging applications stem from using the fiber web as a photodetector. Its ability to allow a large amount of light to pass while still detecting an intensity distribution allows for acquisition of intensity distributions at multiple points along a ray from the source. This ability allows for three-dimensional imaging and for acquisition of phase information, both not possible using conventional detectors.

5.1 Imaging with a fiber web

Because the fibers integrate the intensity over the length of the fiber, imaging with a fiber web can be difficult. One intensity distribution alone is not enough information to properly recover an accurate image. Remember that the fiber conductance depends not only on the area of incident light, but also the intensity of the light at each point. It is this overall power at each point that we are trying to recover with the fiber web imaging system. Knowing the fiber outputs and assuming that there are now at least two points illuminated on the fiber web, it would be impossible to the intensity at each point. For example, Fig. 5-1(a) and Fig. 5-1(b) will have identical outputs if the power of the incident beams is identical. Fig. 5-1(c) will also have the same output as the other two if the power of each point is halved.



(a) (b) (c)

Figure 5-1: Three illumination patterns all producing an ambiguous output. The power of the incident beams in (c) are half of those in (a) and (b).

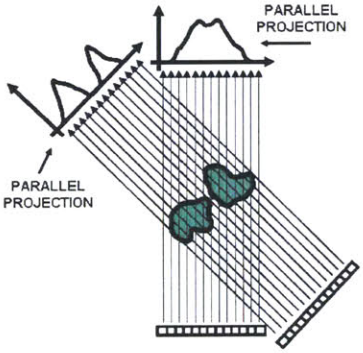


Figure 5-2: Parallel projections of an arbitrary intensity distribution at different angles.

5.1.1 Recovering an image using CAT algorithms

X-ray tomography uses an array of single point emitters and detectors placed in parallel on opposite sides of an object that measure the integral of attenuation as x-ray photons pass through the object. The resulting measurement, deemed a *parallel projection,*[7] can be seen in Fig 5-2. The point emitter/detector pairs correspond to a single fiber in the fiber web. Fig 5-3 shows the measured parallel projection of an incident letter E upon a fiber web. Although the functionality of detection differs between these two detectors, the resulting parallel projections are analogous. This allows the use of the backprojection CAT algorithm in processing the parallel projections to recover the intensity distribution incident upon a fiber web.

The backprojection algorithm uses data acquired at multiple angles as the x-ray emitter and detector are simultaneously rotated about the object. A fiber web allows for multiple solutions to the rotation issue: rotation of the fiber web, rotation of the

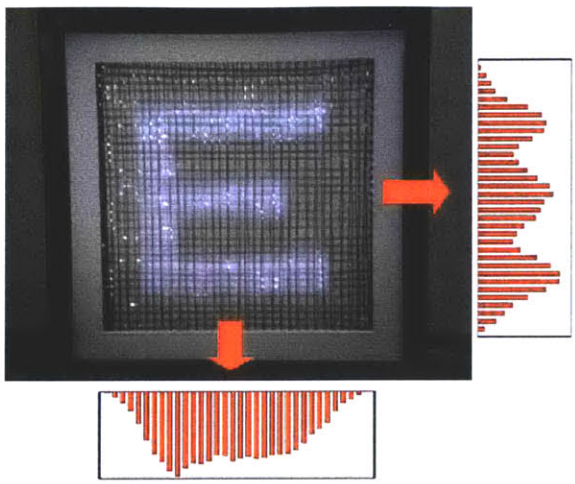


Figure 5-3: Normalized orthogonal parallel projections measured using a fiber web for an incident letter E.

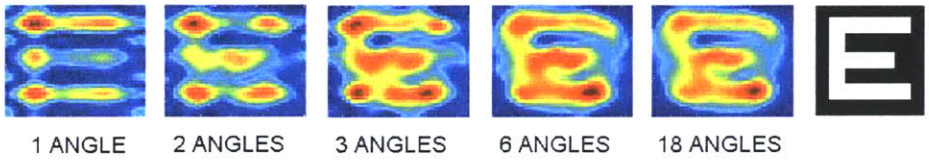


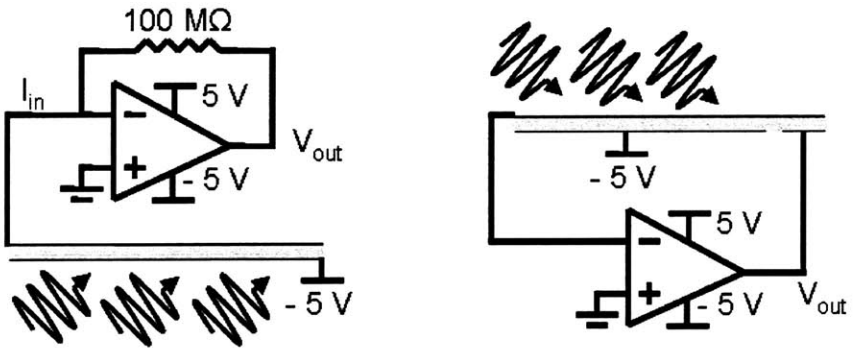
Figure 5-4: Images recovered using data from a varied number of acquisition angles.

imaged object, or using multiple fiber webs offset at fixed angles. This last solution is made possible by the largely transparent nature of the fiber web. It is also significant because the slow, physical rotation of the detector is eliminated allowing for rapid acquisition of data. The results of using the backprojection algorithm with data from a varied number of rotations can be seen in Fig 5-4.

Note that the orthogonal detectors in the fiber web reduces the required rotation of the detector/image to obtain a full data set from 180 rotation required of the x-ray detectors to only a 90 rotation for the fiber web. This property of the fiber web yields a two times higher effective number of acquired data with respect to the number of rotations.

5.1.2 Phase retrieval

Unlike conventional photodetectors, an intensity distribution projected onto a fiber web can pass through the detector with little loss. This enables a second fiber web



(a) (b)

Figure 5-5: Transimpedance amplifier (a) with traditional feedback resistor and (b) a short section of fiber as the feedback resistor.

to be placed a fixed distance behind the first web, both able to recover an incident intensity distribution. As before, the data must be taken at multiple angles of rotation and processed by the backprojection algorithm. The detected intensity distributions can then be analyzed by a phase-retrieval algorithm to reconstruct the image projected onto the arrays. Detailed discussion of this topic is beyond the scope of this thesis, but it is important to grasp the versatility of a fiber web.

5.2 Future work

A characteristic of the fiber that could be of benefit is a short piece of fiber has a higher impedance than a longer piece. Therefore, the large 100 MΩ resistor that is used in the transimpedance amplifier of Fig. 5-5(a) could be replaced by a short section of fiber as in Fig. 5-5(b). This section of fiber could be part of the same fiber that is used as the detector, significantly reducing the number of components on the PCB.

The current fiber web is restricted to 6.4 kbps data rate using the PIC16C745 microcontroller and the HIDCOMM Active X control. Microchip has released a USB 2.0 compliant device, the 18F2455 series which does restrict the device to operate in the HID mode. This microcontroller will allow increased data rates from the fiber web allowing for larger webs with faster data refresh.

Another mode of data transmission that will increase the data rate is wireless communication. Bluetooth has been a growing technology with single-chip transceivers becoming readily available to consumers. A competitor in USB wireless has also emerged from the market and these two options will be explored in detail to find an answer that bets fits our requirements.

Although we were able to demonstrate a three-dimensional detector with two fiber webs, the creation of a single detector is appealing. This can be accomplished by the creation of a sphere detector. Such a detector structure will allow for three-dimensional detection of an incident beam from virtually any angle.

Chapter 6

Conclusion

We were able to construct two-and three-dimensional fiber webs from one-dimensional fibers. The resulting detectors were able to be used in a variety of imaging applications including point detection, vector detection, imaging, and phase retrieval. The large scale of the fiber web enabled image recovery where conventional detectors would have required the use of a lens. Fiber webs are not limited to planar detectors; a spherical detector, for example, can be constructed using the flexible, optoelectronic fibers.

Appendix A

Program code

A.1 Microcontroller embedded program

A.1.1 Acquisition functions

```
#include <16C745.H>

/* no watchdog timer, no code protect, yes power up timer */
#device ADC=8 *=16 // 8-bit ADC 16-bit
#fuses HS, NOWDT, NOPROTECT, PUT

/* tell compiler that clk is 24MHz (use H4 fuse, HS with Rev 1.1) */
#use Delay(Clock=24000000)

/* set port directions manually */

#use fast_io(C)
#use fast_io(B)

/* rs232 serial set up */
#use RS232(BAUD=19200, XMIT=PIN_C6, RCV=PIN_C7)

#define USB_HID_DEVICE TRUE
#define USB_RUN_WHEN_CONFIGURED TRUE

#define USB_EP1_TX_ENABLE 1
#define USB_EP1_TX_SIZE 8 // must change usb_desc report count

#define USB_EP1_RX_ENABLE 1
#define USB_EP1_RX_SIZE 2

#define USB_EP3_TX_ENABLE 1 // added to try and have multiple arrays connected to same comp
#define USB_EP3_TX_SIZE 8

#define USB_EP3_RX_ENABLE 1
#define USB_EP3_RX_SIZE 2

#include <usb.h>
#include <usb_desc.h>
```

```
#include <pic_usb.h>
#include <usb.c>

#define ARRAY_SIZE 64 // 16 was old size

/* Direction of input/output pins
      A0 = 1 = Analog input Rows
      A1 = 1 = Analog input Columns
      A2 = 1 = Analog input
      A3 = 1 = Analog input

      B0 = 0 = Multiplexer Select bit
      B1 = 0 = Multiplexer Select bit
      B2 = 0 = Multiplexer Select bit
      B3 = 0 = Multiplexer Select bit

      C0 = 0 = TEST LED
      C6 = 0 = RS-232 Tx
      C7 = 1 = RS-232 Rx
*/

#define TRIS_A 0b00111111
#define TRIS_B  0b00000000
#define TRIS_C  0b01000000

#define MUX00  0b00000000
#define MUX01 0b00000001
#define MUX02 0b00000010
#define MUX03 0b00000011
#define MUX04 0b00000100
#define MUX05 0b00000101
#define MUX06 0b00000110
#define MUX07 0b00000111
#define MUX08 0b00001000
#define MUX09 0b00001001
#define MUX10 0b00001010
#define MUX11 0b00001011
#define MUX12 0b00001100
#define MUX13 0b00001101
#define MUX14 0b00001110
#define MUX15 0b00001111

void adcsel(int y)
{
      int x;
      switch(y)
      {
            case(0):      {
                  set_adc_channel(1);
                  x = read_adc();
                  break;
            }
            case(1):      {
                  set_adc_channel(0);
                  x = read_adc();
                  break;
            }
            case(2):      {
                  set_adc_channel(2);
                  x = read_adc();
                  break;
```

```
            }
            case(3):    {
                  set_adc_channel(3);
                  x = read_adc();
                  break;
            }
      }
}

void muxsel(int x)
{
      switch(x)
      {
            case(0):    {
                  output_b(MUX00);
                  break;
            }
            case(1):    {
                  output_b(MUX01);
                  break;
            }
            case(2):    {
                  output_b(MUX02);
                  break;
            }
            case(3):    {
                  output_b(MUX03);
                  break;
            }
            case(4):    {
                  output_b(MUX04);
                  break;
            }
            case(5):    {
                  output_b(MUX05);
                  break;
            }
            case(6):    {
                  output_b(MUX06);
                  break;
            }
            case(7):    {
                  output_b(MUX07);
                  break;
            }
            case(8):    {
                  output_b(MUX08);
                  break;
            }
            case(9):    {
                  output_b(MUX09);
                  break;
            }
            case(10):   {
                  output_b(MUX10);
                  break;
            }
            case(11):         {
                  output_b(MUX11);
                  break;
            }
```

```
            case(12):           {
                  output_b(MUX12);
                  break;
            }                                                                    160
            case(13):           {
                  output_b(MUX13);
                  break;
            }
            case(14):           {
                  output_b(MUX14);
                  break;
            }
            case(15):           {
                  output_b(MUX15);                                               170
                  break;
            }
      }
}

void sample_data(int samples)
{
      int i,j,k; //i sets mux address, j sets adc channel
      int8 out_data_a[USB_EP1_TX_SIZE]; //int8 out_data[ARRAY_SIZE];
      int8 out_data_b[USB_EP1_TX_SIZE];                                          180
      int8 out_data_c[USB_EP1_TX_SIZE];
      int8 out_data_d[USB_EP1_TX_SIZE];
      int8 out_data_e[USB_EP1_TX_SIZE];
      int8 out_data_f[USB_EP1_TX_SIZE];
      int8 out_data_g[USB_EP1_TX_SIZE];
      int8 out_data_h[USB_EP1_TX_SIZE];

      int usb_delay = 10;
      int avg_size = 5; //number of samples to average over
                                                                                 190
      for (k=0; k<avg_size; k++)
      {
            for (j=0; j<4; j++)
            {
                  adcsel(j); //set adc channel, order is 1,0,2,3
                  delay_us(100);
                  for (i=0; i<16; i++)
                  {
                        muxsel(i);
                        delay_us(10);                                            200
                        //32x32 Loop
                        if (j < 2)
                        {
                              if (i < 4)
                              {
                                    out_data_a[i+j*4] = read_adc();
                              }
                              else if (i < 8)
                              {
                                    out_data_b[i-4+j*4] = read_adc();            210
                              }
                              else if (i < 12)
                              {
                                    out_data_c[i-8+j*4] = read_adc();
                              }
                              else
                              {
```

```
                        out_data_d[i-12+j*4] = read_adc();
                    }
                }
                else //j = 2 and j = 3
                {
                    if (i < 4)
                    {
                        out_data_e[i+(j-2)*4] = read_adc();
                    }
                    else if (i < 8)
                    {
                        out_data_f[i-4+(j-2)*4] = read_adc();
                    }
                    else if (i < 12)
                    {
                        out_data_g[i-8+(j-2)*4] = read_adc();
                    }
                    else
                    {
                        out_data_h[i-12+(j-2)*4] = read_adc();
                    }
                }
            }
        }
        // put data into large temp buffer

    }

    usb_put_packet(1, (out_data_a), USB_EP1_TX_SIZE, TOGGLE);
    delay_ms(usb_delay);
    usb_put_packet(1, (out_data_b), USB_EP1_TX_SIZE, TOGGLE);
    delay_ms(usb_delay);
    usb_put_packet(1, (out_data_c), USB_EP1_TX_SIZE, TOGGLE);
    delay_ms(usb_delay);
    usb_put_packet(1, (out_data_d), USB_EP1_TX_SIZE, TOGGLE);
    delay_ms(usb_delay);
    usb_put_packet(1, (out_data_e), USB_EP1_TX_SIZE, TOGGLE);
    delay_ms(usb_delay);
    usb_put_packet(1, (out_data_f), USB_EP1_TX_SIZE, TOGGLE);
    delay_ms(usb_delay);
    usb_put_packet(1, (out_data_g), USB_EP1_TX_SIZE, TOGGLE);
    delay_ms(usb_delay);
    usb_put_packet(1, (out_data_h), USB_EP1_TX_SIZE, TOGGLE);
    delay_ms(usb_delay);
}


void main()
{
    int8 in_data[USB_EP1_RX_SIZE];
    int i;

    printf("\r\n\r\nPCM: v"); //Version Number
    printf(__PCM__);

    set_tris_a(TRIS_A);
    set_tris_b(TRIS_B);
    set_tris_c(TRIS_C);
    setup_adc(ADC_CLOCK_DIV_32);//setup_adc(ADC_CLOCK_INTERNAL);
    setup_adc_ports(ALL_ANALOG);
  set_adc_channel(1); // ADC channel will be selected later, order 1,0,2,3
```

```
   usb_init();

   // printf("\r\n\r\nWaiting for enumeration...");

   usb_wait_for_enumeration();
printf("\r\n\r\n***Enumerated***\r\n");

START:
while(usb_enumerated())
{
        #IF USB_RUN_WHEN_CONFIGURED
   // if (usb_kbhit(1)) //checks status of RX buffer, true = data
   //{
        usb_gets(1, in_data, USB_EP1_RX_SIZE); // Waits for data in the buffer
   printf("\r\n--> Received 2 bytes: 0x%X 0x%X",in_data[0],in_data[1]);
   //}
   //if (in_data[0]==10 && in_data[1]==255) // calibration proceedure Array #1
   if (in_data[0]==100 && in_data[1]==200) // calibration proceedure Array #2
   {
        for (i=0; i<20; i++)
        {
             sample_data(ARRAY_SIZE);
          }
        }
        else
        {
             goto RUN;
        }

   WAIT:
   usb_gets(1, in_data, USB_EP1_RX_SIZE);  // Wait for host to act
   RUN:
   //if (in_data[0]==101 & in_data[1]==1) // Array #1
   if (in_data[0]==55 & in_data[1]==90) // Array #2
   {
        for (i=0; i<5; i++)
        {
             sample_data(ARRAY_SIZE);
          }
          goto WAIT;
        }
        //else if (in_data[0]==211 & in_data[1]==112) // Sample 1 time for Vector Prog Array #1
        else if (in_data[0]==79 & in_data[1]==84) // Array #2
        {
             while(TRUE)
             {
                  sample_data(ARRAY_SIZE);
                  if (usb_kbhit(1))
             {
                  GOTO START;
               }
             }
        }
        else
        {
             goto START;
        }
   #ENDIF
   }
printf("\r\n\r\nDevice Un-configured.r\n");
```

```
}

// Move the 0-20 loop into the sampel_data function so that data is more reliable but speed is
// not sacrificed.
```

A.1.2 USB Descriptor

```
////////////////////////////////////////////////////////////////////////////
///                    usb_desc.h                                       ////
////                                                                    ////
//// An example set of device / configuration descriptors.            ////
////                                                                    ////
//// Descriptors are application dependent (especially HID) so you may ////
//// have to edit them.                                                 ////
////                                                                    ////
//// This file is part of CCS's PIC USB driver code, which includes:   ////
////   usb_desc.h - an example set of config and device descriptors    ////
////   usb.c - USB token and request handler code                      ////
////   usb.h - definitions, prototypes and global variables            ////
////   And the hardware peripheral level.  At the time of writing,     ////
////    CCS provides two hardware peripheral drivers.  National        ////
////    USBN960x (usbn960x.cc) and PIC16C7x5 (usb_pic.c).              ////
////                                                                   ////
//// Two examples are given using CCS's PIC USB driver                ////
//// ex_usb_scope.c is a bulk device that sends 512 bytes to the      ////
//// host and uses the USBN960x.  ex_usb_hid.c is a HID device and     ////
//// uses the PIC16C7x5.                                              ////
////                                                                    ////
////////////////////////////////////////////////////////////////////////////
////                                                                    ////
//// Version History:                                                 ////
////                                                                    ////
//// May 6th, 2003: Fixed non-HID descriptors pointing to faulty      ////
////             strings                                             ////
////                                                                    ////
//// August 2nd, 2002: Initial Public Release                         ////
////                                                                    ////
////                                                                    ////
////////////////////////////////////////////////////////////////////////////
////       (C) Copyright 1996,2002 Custom Computer Services          ////
//// This source code may only be used by licensed users of the CCS    ////
//// C compiler.  This source code may only be distributed to other    ////
//// licensed users of the CCS C compiler.  No other use,             ////
//// reproduction or distribution is permitted without written        ////
//// permission.  Derivative programs created using this software     ////
//// in object code form are not restricted in any way.               ////
////////////////////////////////////////////////////////////////////////////

#IFNDEF __USB_DESCRIPTORS__
#DEFINE __USB_DESCRIPTORS__

#include "usb.h"

#IF USB_HID_DEVICE
  //usb descriptor reports
  #DEFINE USB_NUM_CONFIGURATIONS 1
  #DEFINE USB_NUM_INTERFACES     1
  #DEFINE USB_NUM_ENDPOINTS      2        //endpoint 1 OUT, endpoint 1 IN.  endpoint 0 doesnt count here
  #DEFINE USB_NUM_CLASSES        1        //hid

  //you cant use pointers when storing constants to program memory.
```

```
BYTE CONST USB_CONFIG_DESCRIPTORS[USB_NUM_CONFIGURATIONS]={0};//0
BYTE CONST USB_INTERFACE_DESCRIPTORS[USB_NUM_INTERFACES]={9};//9
BYTE CONST USB_CLASS_DESCRIPTORS[USB_NUM_CLASSES]={18};//18
BYTE CONST USB_ENDPOINT_DESCRIPTORS[USB_NUM_ENDPOINTS]={27,34};//27

#DEFINE USB_TOTAL_CONFIG_LEN      41  //config+interface+class+endpoint+endpoint (2 endpoints)

#DEFINE USB_HID_DESC_LEN          28

BYTE CONST USB_DEVICE_DESC[] ={
  //starts of with device configuration. only one possible
    USB_DEVICE_DESC_LEN, //the length of this report   ==1
    0x01, //the constant DEVICE (DEVICE 0x01)  ==2
    0x10,0x01, //usb version in bcd (pic167xx is 1.1) ==3,4
    0x00, //class code ==5
    0x00, //subclass code ==6
    0x00, //protocol code ==7
    USB_MAX_EP0_PACKET_LENGTH, //max packet size for endpoint 0. (SLOW SPEED SPECIFIES 8) ==8
    0x61,0x04, //0x60,0x04, //vendor id (0x04D8 is Microchip, or is it 0x0461 ??)
    0x20,0x01, //0x20,0x00, //product id   ==11,12  //don't use ffff says usb-by-example guy.  oops
    0x01,0x02, //0x01,0x01, //device release number  ==13,14
    0x01, //index of string description of manufacturer. therefore we point to string_1 array (see below)  ==15
    0x02, //index of string descriptor of the product  ==16
    0x00, //index of string descriptor of serial number  ==17
    USB_NUM_CONFIGURATIONS  //number of possible configurations  ==18
};


BYTE CONST USB_CONFIG_DESC[USB_TOTAL_CONFIG_LEN] = {
//IN ORDER TO COMPLY WITH WINDOWS HOSTS, THE ORDER OF THIS ARRAY MUST BE:
  //   config(s)
  //   interface(s)
  //   class(es)
  //   endpoint(s)

//config_descriptor for config index 1
    USB_CONFIG_DESC_LEN, //length of descriptor size        ==1
    USB_CONFIG_DESC_KEY, //constant CONFIGURATION (CONFIGURATION 0x02)    ==2
    USB_TOTAL_CONFIG_LEN,0, //size of all data returned for this config     ==3,4
    USB_NUM_INTERFACES, //number of interfaces this device supports      ==5
    0x01, //identifier for this configuration.  (IF we had more than one configurations)     ==6
    0x00, //index of string descriptor for this configuration     ==7
    0xC0, //bit 6=1 if self powered, bit 5=1 if supports remote wakeup (we don't), bits 0-4 unused and bit7=1        ==8
    0x32, //maximum bus power required (maximum milliamperes/2)  (0x32 = 100mA)

//interface descriptor 1
    USB_INTERFACE_DESC_LEN, //length of descriptor      =10
    USB_INTERFACE_DESC_KEY, //constant INTERFACE (INTERFACE 0x04)       =11
    0x00, //number defining this interface (IF we had more than one interface)    ==12
    0x00, //alternate setting     ==13
    USB_NUM_ENDPOINTS, //number of endpoins, except 0 (pic167xx has 3, but we dont have to use all).       ==14
    0x03, //class code, 03 = HID     ==15
    0x00, //subclass code //boot     ==16
    0x00, //protocol code      ==17
    0x00, //index of string descriptor for interface      ==18

//class descriptor 1  (HID)
    USB_CLASS_DESC_LEN, //length of descriptor    ==19
    USB_CLASS_DESC_KEY, //dscriptor type (0x21 == HID)      ==20
    0x00,0x01, //hid class release number (1.0) (try 1.10)      ==21,22
    0x00, //localized country code (0 = none)       ==23
```

```
    0x01, //number of hid class descrptors that follow (1)      ==24
    0x22, //report descriptor type (0x22 == HID)               ==25
    USB_HID_DESC_LEN, 0x00, //length of report descriptor          ==26,27

//endpoint descriptor                                                          120
    USB_ENDPOINT_DESC_LEN, //length of descriptor                  ==28
    USB_ENDPOINT_DESC_KEY, //constant ENDPOINT (ENDPOINT 0x05)        ==29
    0x81, //endpoint number and direction (0x81 = EP1 IN)       ==30
    0x03, //transfer type supported (0x03 is interrupt)         ==31
    0x08,0x00, //maximum packet size supported                  ==32,33
    10,  //polling interval, in ms.  (cant be smaller than 10)     ==34

//endpoint descriptor
    USB_ENDPOINT_DESC_LEN, //length of descriptor                  ==35
    USB_ENDPOINT_DESC_KEY, //constant ENDPOINT (ENDPOINT 0x05)        ==36     130
    0x01, //endpoint number and direction (0x01 = EP1 OUT)       ==37
    0x03, //transfer type supported (0x03 is interrupt)         ==38
    0x08,0x00, //maximum packet size supported                  ==39,40
    10 //polling interval, in ms.  (cant be smaller than 10)    ==41
};


//////////////////////////////////////////////////////////////////////
///
///  HID Report.  Tells HID driver how to handle and deal with                 140
///  received data.  HID Reports can be extremely complex,
///  see HID specifcation for help on writing your own.
///
///  CCS example uses a vendor specified usage, that sends and
///  receives 2 absolute bytes ranging from 0 to 0xFF.
///
//////////////////////////////////////////////////////////////////////
BYTE CONST USB_HID_DESC[] = {  //len=28
  6, 0, 255,    // Usage Page = Vendor Defined
  9, 1,          // Usage = IO device                                          150
  0xa1, 1,      // Collection = Application
  0x19, 1,       // Usage minimum
  0x29, 8,       // Usage maximum

  0x15, 0x80,       // Logical minimum (-128)
  0x25, 0x7F,       // Logical maximum (127)

  0x75, 8,       // Report size = 8 (bits)
  0x95, 8,       // Report count = 16 bits (2 bytes), changed to 8 bytes
  0x81, 2,       // Input (Data, Var, Abs)                                     160
  0x19, 1,       // Usage minimum
  0x29, 8,       // Usage maximum
  0x91, 2,       // Output (Data, Var, Abs)
  0xc0           // End Collection
};

#ELSE

  //usb descriptor reports
  #DEFINE USB_NUM_CONFIGURATIONS 1                                           170
  #DEFINE USB_NUM_INTERFACES     1
  #DEFINE USB_NUM_ENDPOINTS      2        //endpoint 1 OUT, endpoint 1 IN.  endpoint 0 doesnt count here
  #DEFINE USB_NUM_CLASSES        0        //hid

  //you cant use pointers when storing constants to program memory.
  BYTE CONST USB_CONFIG_DESCRIPTORS[USB_NUM_CONFIGURATIONS]={0};//0
```

```
BYTE CONST USB_INTERFACE_DESCRIPTORS[USB_NUM_INTERFACES]={9};//9
BYTE CONST USB_ENDPOINT_DESCRIPTORS[USB_NUM_ENDPOINTS]={18,25};//27

#DEFINE USB_TOTAL_CONFIG_LEN      32 //config+interface+endpoint+endpoint (2 endpoints)

//device descriptor
BYTE CONST USB_DEVICE_DESC[USB_DEVICE_DESC_LEN] ={
    USB_DEVICE_DESC_LEN,          //the length of this report
    0x01,              //constant DEVICE (0x01)
    0x10,0x01,         //usb version in bcd
    0x00,              //class code (if 0, interface defines class.  FF is vendor defined)
    0x00,              //subclass code
    0x00,              //protocol code
    USB_MAX_EP0_PACKET_LENGTH,   //max packet size for endpoint 0. (SLOW SPEED SPECIFIES 8)
    0x61,0x04,         //vendor id (0x04D8 is Microchip)
    0x04,0x01,         //product id
    0x01,0x02,         //device release number
    0x01,              //index of string description of manufacturer. therefore we point to string_1 array (see below)
    0x02,              //index of string descriptor of the product
    0x00,              //index of string descriptor of serial number
    USB_NUM_CONFIGURATIONS   //number of possible configurations
};

//configuration descriptor
BYTE CONST USB_CONFIG_DESC[USB_TOTAL_CONFIG_LEN] = {
//config_descriptor for config index 1
    USB_CONFIG_DESC_LEN,     //length of descriptor size
    USB_CONFIG_DESC_KEY,          //constant CONFIGURATION (0x02)
    USB_TOTAL_CONFIG_LEN,0,  //size of all data returned for this config
    USB_NUM_INTERFACES,      //number of interfaces this device supports
    0x01,              //identifier for this configuration.  (IF we had more than one configurations)
    0x00,              //index of string descriptor for this configuration
    0xC0,              //bit 6=1 if self powered, bit 5=1 if supports remote wakeup (we don't), bits 0-4 reserved and bit7=1
    0x32,              //maximum bus power required (maximum milliamperes/2)  (0x32 = 100mA)

//interface descriptor 0 alt 0
    USB_INTERFACE_DESC_LEN,  //length of descriptor
    USB_INTERFACE_DESC_KEY,      //constant INTERFACE (0x04)
    0x00,              //number defining this interface (IF we had more than one interface)
    0x00,              //alternate setting
    USB_NUM_ENDPOINTS,       //number of endpoints, except 0.
    0xFF,              //class code, FF = vendor defined
    0xFF,              //subclass code, FF = vendor
    0xFF,              //protocol code, FF = vendor
    0x00,              //index of string descriptor for interface

//endpoint descriptor
    USB_ENDPOINT_DESC_LEN, //length of descriptor
    USB_ENDPOINT_DESC_KEY,     //constant ENDPOINT (0x05)
    0x81,             //endpoint number and direction (0x81 = EP1 IN)
    0x02,             //transfer type supported (0 is control, 1 is iso, 2 is bulk, 3 is interrupt)
    0x40,0x00,         //maximum packet size supported
    0x01,             //polling interval in ms. (for interrupt transfers ONLY)

//endpoint descriptor
    USB_ENDPOINT_DESC_LEN, //length of descriptor
    USB_ENDPOINT_DESC_KEY,     //constant ENDPOINT (0x05)
    0x01,             //endpoint number and direction (0x01 = EP1 OUT)
    0x02,             //transfer type supported (0 is control, 1 is iso, 2 is bulk, 3 is interrupt)
    0x40,0x00,         //maximum packet size supported
```

```
      0x01,           //polling interval in ms. (for interrupt transfers ONLY)

 };
#ENDIF


//////////////////////////////////////////////////////////////////////
///
///  start string descriptors
///  String 0 is a special language string, and must be defined.  People in U.S.A. can leave this alone.
///
///  You must define the length else get_next_string_character() will not see the string
///  Current code only supports 10 strings (0 thru 9)
///
//////////////////////////////////////////////////////////////////////

#DEFINE USB_STRING_0_LEN 4
BYTE CONST USB_STRING_0[4] = {  //string index 0 is special
       USB_STRING_0_LEN, //length of string index
       0x03, //descriptor type 0x03 (STRING)
       0x09,0x04   //Microsoft Defined for US-English
};

#DEFINE USB_STRING_1_LEN 8
BYTE CONST USB_STRING_1[8] = { //we defined string index 1 earlier as manuf string
       USB_STRING_1_LEN, //length of string index
       0x03, //descriptor type 0x03 (STRING)
       'T',0,
       'W',0,
       'O',0
};

#IF USB_HID_DEVICE
  #DEFINE USB_STRING_2_LEN 26
  BYTE CONST USB_STRING_2[26] = {  //we defined string index 2 as description of prodcut
         USB_STRING_2_LEN, //length of string index
         0x03, //descriptor type 0x03 (STRING)
         'R',0,
         'L',0,
         'E',0,
         ' ',0,
         '3',0,
         '2',0,
         'x',0,
         '3',0,
         '2',0,
         ' ',0,
         '2',0,
         ' ',0
  };
#else
  #DEFINE USB_STRING_2_LEN 28
  BYTE CONST USB_STRING_2[28] = {  //we defined string index 2 as description of prodcut
         USB_STRING_2_LEN, //length of string index
         0x03, //descriptor type 0x03 (STRING)
         'C',0,
         'C',0,
         'S',0,
         ' ',0,
         'B',0,
         'u',0,
```

```
        'l',0,
        'k',0,
        ' ',0,
        'D',0,
        'e',0,
        'm',0,
        'o',0
  };
#endif

#ENDIF
```

A.2 3D Vector Detection

```
Option Strict Off
Option Explicit On
Imports System
Imports System.Drawing

Friend Class frm3D
    Inherits System.Windows.Forms.Form
    Dim Stp, Initialized As Boolean
    Dim OutBuff1(), OutBuff2(), Buffer1(), Buffer2() As Byte
    Dim CalBuff1(), MainBuff1(), rowThresh1, colThresh1 As Single 'Calibration values
    Dim CalBuff2(), MainBuff2(), rowThresh2, colThresh2 As Single
    Dim ArraySize, BufferSize, SampleSize, stepSize As Integer
    Dim rowLoc1, colLoc1, rowLoc2, colLoc2 As Integer

    Public pictureBox1 As New PictureBox 'Background 1

    Private Sub Graph'Closed(ByVal eventSender As System.Object, ByVal eventArgs As System.EventArgs) Handles MyBase.Closed
        Stp = True
        AxHIDComm1.Disconnect()
        AxHIDComm2.Disconnect()
        AxHIDComm1.Uninit()
        AxHIDComm2.Uninit()
        System.Windows.Forms.Application.DoEvents()
    End Sub

    Private Sub Start'Click(ByVal eventSender As System.Object, ByVal eventArgs As System.EventArgs) Handles Start.Click
        Dim i, j, x, y, rowOffset1, colOffset1, rowOffset2, colOffset2 As Integer
        Dim maxRow1, maxCol1, maxRow2, maxCol2 As Integer
        Dim color1, color2 As Color
        Dim avgsize As Integer

        'Dim arrData(8, 1)
        ReDim Buffer1(7)
        ReDim Buffer2(7)

        ArraySizeComboBox.Enabled = False
        Start.Enabled = False
        Stop'Renamed.Enabled = True
        Stp = False
        avgsize = 5


        'Raw
        OutBuff1(0) = 211
        OutBuff1(1) = 112
        AxHIDComm1.WriteTo(OutBuff1, 2)
```

```
'Raw
OutBuff2(0) = 79
OutBuff2(1) = 84
AxHIDComm2.WriteTo(OutBuff2, 2)

Do While Stp = False
   'Initialize variables for each loop
   rowThresh1 = cmbThresh1.SelectedItem
   colThresh1 = cmbThresh1.SelectedItem
   rowThresh2 = cmbThresh2.SelectedItem
   colThresh2 = cmbThresh2.SelectedItem

   maxRow1 = 0 'Set to 0 each time, later check to see if maxRow != 0
   maxCol1 = 0 '
   maxRow2 = 0
   maxCol2 = 0
   rowLoc1 = 0
   colLoc1 = 0
   rowLoc2 = 0
   colLoc2 = 0

   ReDim Buffer1(7)
   ReDim Buffer2(7)
   ReDim MainBuff1(2 * ArraySize - 1)
   ReDim MainBuff2(2 * ArraySize - 1)

   For x = 1 To avgsize
      For y = 0 To 7
         'Collect all the data
         Buffer1 = AxHIDComm1.ReadFrom(BufferSize)
         If BufferSize < 8 Then Exit Sub
         Buffer2 = AxHIDComm2.ReadFrom(BufferSize)
         If BufferSize < 8 Then Exit Sub
         For i = 0 To 7
            'MainBuff1(i + y * 8) = Buffer1(i) 'Used for single point
            'MainBuff2(i + y * 8) = Buffer2(i)
            MainBuff1(i + y * 8) += (Buffer1(i) / avgsize) ' * CalBuff1(i + y * 8)))
            MainBuff2(i + y * 8) += (Buffer2(i) / avgsize) '* CalBuff2(i + y * 8)))
         Next i
      Next y
      MainBuff2(3) = 0
      MainBuff2(27) = 0
      MainBuff2(31) = 0
   Next x

   'Use when plotting single point
   For i = 0 To ArraySize - 1 'Look for the largest deviation
      MainBuff1(i) = MainBuff1(i) / CalBuff1(i) '- 1 'Normalize then subtract 1
      MainBuff2(i) = MainBuff2(i) / CalBuff2(i)
      MainBuff1(i + ArraySize) = MainBuff1(i + ArraySize) / CalBuff1(i + ArraySize)
      MainBuff2(i + ArraySize) = MainBuff2(i + ArraySize) / CalBuff2(i + ArraySize)

      If MainBuff1(i) > rowThresh1 Then
         rowThresh1 = MainBuff1(i)
         maxRow1 = i 'convert to 8-1 from 0-7, added "8-"to reverse data
         rowLoc1 = ArraySize - 1 - i 'rowLoc1 = 7 - i
      End If

      If MainBuff2(i) > rowThresh2 Then
         rowThresh2 = MainBuff2(i)
         maxRow2 = i 'convert to 1-8 from 0-7
```

```
            rowLoc2 = ArraySize - 1 - i 'rowLoc2 = 7 - i
        End If
                                                                                    110
        If MainBuff1(i + ArraySize) > colThresh1 Then
            colThresh1 = MainBuff1(i + ArraySize)
            maxCol1 = i
            colLoc1 = ArraySize - 1 - i 'colLoc1 = i
        End If
        If MainBuff2(i + ArraySize) > colThresh2 Then
            colThresh2 = MainBuff2(i + ArraySize)
            maxCol2 = i
            colLoc2 = ArraySize - 1 - i 'colLoc2 = i
        End If                                                                      120
    Next i

    Label1.Text = CStr(rowThresh1)
    Label2.Text = CStr(colThresh1)

    Me.pictureBox1.Refresh()

    System.Windows.Forms.Application.DoEvents() 'Refresh screen
  Loop
                                                                                    130
  'Send Data to the PIC to reset the loop to the calibrate proceedure
  OutBuff1(0) = 255
  OutBuff1(1) = 255
  AxHIDComm1.WriteTo(OutBuff1, 2)
  AxHIDComm2.WriteTo(OutBuff1, 2)

  Start.Enabled = False
  Calibrate.Enabled = True
  Stop'Renamed.Enabled = False
                                                                                    140

End Sub

Private Sub Stop'Renamed'Click(ByVal eventSender As System.Object, ByVal eventArgs As System.EventArgs)
  Start.Enabled = False
  Stop'Renamed.Enabled = False
  Calibrate.Enabled = True
  Stp = True

  OutBuff1(0) = 255                                                                 150
  OutBuff1(1) = 255
  AxHIDComm1.WriteTo(OutBuff1, 2)
  AxHIDComm2.WriteTo(OutBuff1, 2)
End Sub

Private Sub Graph'Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
  Dim picloc As Point
  Dim x As Image
  'Dim picturebox0 As PictureBox
                                                                                    160
  Stop'Renamed.Enabled = False
  Start.Enabled = False
  Initialized = False

  ArraySizeComboBox.SelectedIndex = 2
  ArraySize = ArraySizeComboBox.SelectedItem

  cmbThresh1.SelectedIndex = 4
```

```
    cmbThresh2.SelectedIndex = 4

    pictureBox1.BackColor = Color.Black
    pictureBox1.Width = 900 '320
    pictureBox1.Height = 600 'pictureBox1.Width
    pictureBox1.Left = 100
    pictureBox1.Top = 50

    SampleSize = 20

    stepSize = pictureBox1.Width / ArraySize

    'pictureBox1.BackColor = Color.Black
    rowLoc1 = 0
    rowLoc2 = 0
    colLoc1 = 0
    colLoc2 = 0

    ' Connect the Paint event of the PictureBox to the event handling method.
    AddHandler pictureBox1.Paint, AddressOf Me.pictureBox1_Paint

    ' Add the PictureBox control to the Form.
    Me.Controls.Add(pictureBox1)

End Sub
Private Sub pictureBox1_Paint(ByVal sender As System.Object, ByVal e As System.Windows.Forms.PaintEventArgs)
    ' Create a local version of the graphics object for the PictureBox.
    Dim g As Graphics = e.Graphics
    Dim i, x, y, xColStep, xRowStep, yStep As Integer
    Dim p As New Pen(Color.White, 5)
    Dim r As New Pen(Color.PaleVioletRed, 18)
    Dim d As New Pen(Color.DarkBlue, 18)
    Dim bX1, bX2, bY1, bY2 As Integer
    Dim fX1, fX2, fY1, fY2 As Integer
    Dim lX1, lX2, lY1, lY2 As Integer

    'Back array points
    bX1 = 480
    bY1 = 20
    bX2 = 715
    bY2 = Int(bX2 - bX1) / 1.25

    'front array points
    fX1 = 0
    fY1 = 20
    fX2 = fX1 + bX2 - bX1
    fY2 = Int(fX2 - fX1) / 1.25 + fY1 - bY1

    xColStep = Int(bX2 - bX1) / ArraySize
    xRowStep = Int(bY2 - bY1) / ArraySize
    yStep = Int(400 - 30) / ArraySize

    'Draw back array
    For i = 0 To 7
        'horizontal
        g.DrawLine(p, bX1, bY1 + 50 * i, bX2, bY2 + 50 * i)
        'vertical
        g.DrawLine(p, bX1 + 10 + 30 * i, 15 + i * 21, bX1 + 10 + 30 * i, 400 + i * 21)
    Next i

    If rowLoc1 > 0 And rowLoc2 > 0 And colLoc1 > 0 And colLoc2 > 0 Then
```

```
        lX1 = colLoc1 * xColStep + fX1
        lY1 = (rowLoc1 - 1) * yStep + fY1 + (colLoc1 - 1) * xRowStep
        lX2 = colLoc2 * xColStep + bX1
        lY2 = (rowLoc2 - 1) * yStep + bY1 + (colLoc2 - 1) * xRowStep

        g.DrawLine(r, lX1, lY1, lX2, lY2)
        'g.DrawEllipse(r, lX1 - 9, lY1 - 9, 18, 18)
        g.DrawEllipse(d, lX2 - 9, lY2 - 9, 18, 18)
    End If

    'Draw Front Array
    For i = 0 To 7
        'horizontal
        g.DrawLine(p, fX1, fY1 + 50 * i, fX2, fY2 + 50 * i)
        'vertical
        g.DrawLine(p, fX1 + 10 + 30 * i, 15 + i * 21, fX1 + 10 + 30 * i, 400 + i * 21)
    Next i

    If rowLoc1 > 0 And rowLoc2 > 0 And colLoc1 > 0 And colLoc2 > 0 Then
        g.DrawEllipse(d, lX1 - 9, lY1 - 9, 18, 18)
    End If

End Sub

Private Sub Calibrate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Calibrate.Click
    ReDim CalBuff1(2 * ArraySize - 1)
    ReDim CalBuff2(2 * ArraySize - 1)
    ReDim OutBuff1(1), OutBuff2(1)
    Dim i, x, y As Integer

    ReDim Buffer1(7) '8 bits 0-7
    ReDim Buffer2(7)
    Initialized = True
    OutBuff1(0) = 10
    OutBuff1(1) = 255
    OutBuff2(0) = 100
    OutBuff2(1) = 200
    BufferSize = 8 '8 buffer spaces

    AxHIDComm1.Connect()
    AxHIDComm1.WriteTo(OutBuff1, 2)
    AxHIDComm2.Connect()
    AxHIDComm2.WriteTo(OutBuff2, 2)

    lblArray1.Text = AxHIDComm1.get_MatchManufacturer
    lblArray2.Text = AxHIDComm2.get_MatchManufacturer

    For x = 1 To SampleSize
        For y = 0 To 7
            Buffer1 = AxHIDComm1.ReadFrom(BufferSize)
            If BufferSize < 7 Then Exit Sub
            Buffer2 = AxHIDComm2.ReadFrom(BufferSize)
            If BufferSize < 7 Then Exit Sub
            For i = 0 To 7
                CalBuff1(i + y * 8) = CalBuff1(i + y * 8) + Buffer1(i)
                CalBuff2(i + y * 8) = CalBuff2(i + y * 8) + Buffer2(i)
            Next i
        Next y
    Next x
    For x = 0 To (2 * ArraySize - 1)
        CalBuff1(x) = CalBuff1(x) / SampleSize + 1 'Normalize for number of samples taken
```

```
            CalBuff2(x) = CalBuff2(x) / SampleSize + 1
        Next x

        Start.Enabled = True
        Calibrate.Enabled = False

    End Sub
    Private Sub ArraySizeComboBox_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)

        ArraySize = ArraySizeComboBox.SelectedItem                                          300

        stepSize = pictureBox1.Width / ArraySize

    End Sub

    Private Sub btnArray1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnArray1.Click
        'AxHIDComm1.ShowPropertyPages()
    End Sub

    Private Sub btnArray2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnArray2.Click 310
        'AxHIDComm2.ShowPropertyPages()
    End Sub

End Class
```

Appendix B

Parts list

Table B.1: Electronics Parts

Mfg. Part #	Digikey Part #	Description
PIC16C745/JW	PIC16C745/JW-ND	IC MCU EPROM8KX14 USB A/D 28CDIP
CD74HC4067M	296-9225-5-ND	IC MUX/DEMUX ANALOG HS 24-SOIC
AD8625AR	AD8625AR-ND	IC OPAMP QUAD JFET R-R 14-SOIC
LM2574M-5.0	LM2574M-5.0-ND	IC REG SIMPLE SWITCHER 14-SOIC
CR75-221KC	308-1253-1-ND	POWER INDUCTOR 220UH 0.49A SMD
T491X227K010AS	399-1582-1-ND	CAPACITOR TANT 220UF 10V 10% SMD
T491D107K010AS	399-1579-1-ND	CAPACITOR TANT 100UF 10V 10% SMD
T491D226K010AS	399-1573-1-ND	CAPACITOR TANT 22UF 10V 10% SMD
T491A104M035AS	399-1644-1-ND	CAPACITOR TANT .10UF 35V 20% SMD
CSTCW24M0X53-R0	490-1205-1-ND	RESONATOR 24.0MHZ CERAMIC
BAT750-7	BAT750DICT-ND	RECT SCHOTTKY 40V 0.75A SOT-23
9C08052A1002FKHFT	311-10.0KCCT-ND	RES 10.0K OHM 1/8W 1% 0805 SMD
9C08052A1501FKHFT	311-1.50KCCT-ND	RES 1.50K OHM 1/8W 1% 0805 SMD
897-30-004-90-000000	ED90003-ND	CONN USB RECEPT TYPE B PCB
AK672/2-2	AE1301-ND	CABLE USB A-B MALE 2M 2.0 VERS
AR28-HZL/7/01-TT	AE7308-ND	IC SOCKET .300 28 DIP GOLD
MOX-200001006J	MOX200J-100M-ND	RES THICK FILM 100M OHM .25W 5%

Appendix C

Full Circuit Schematic

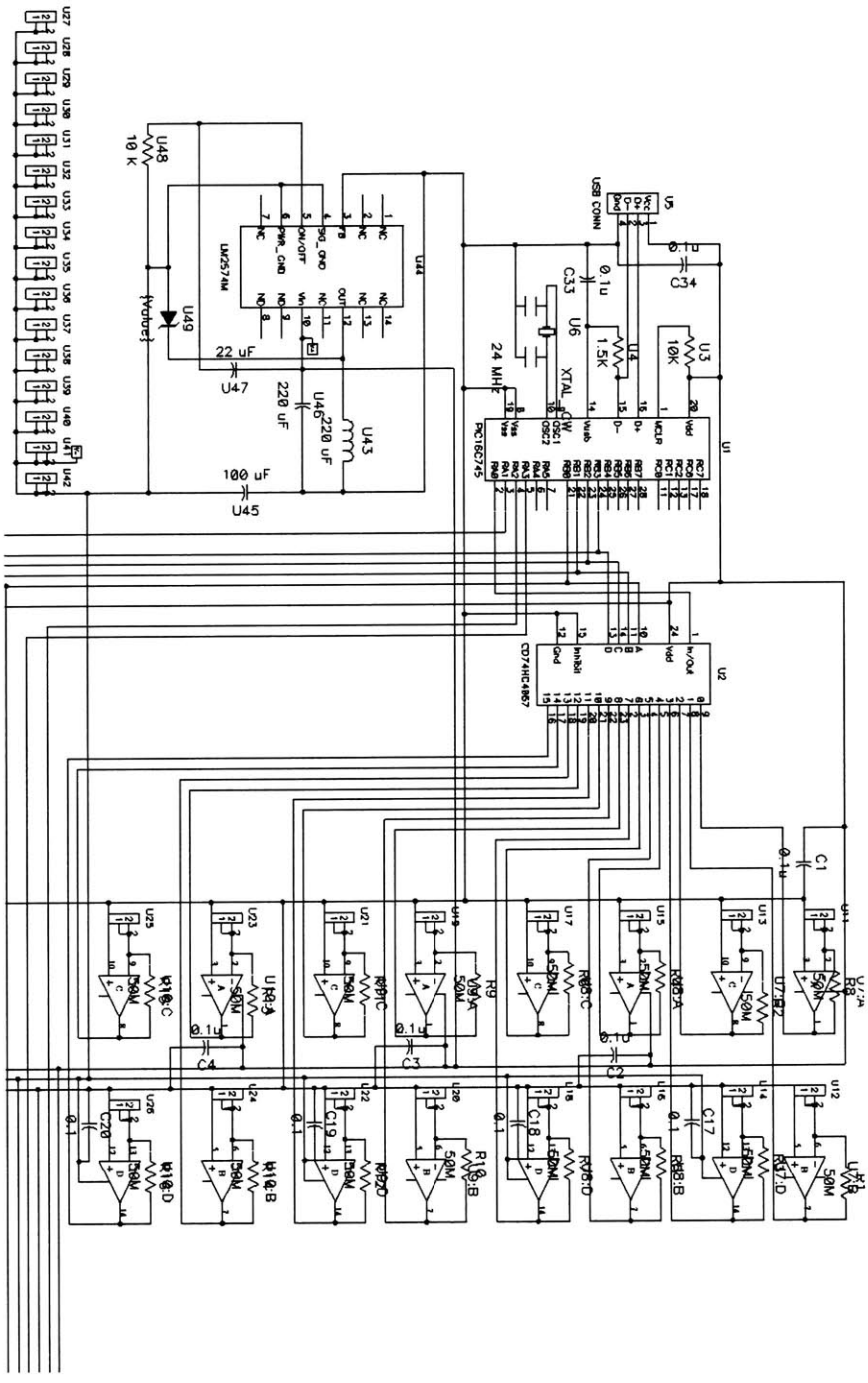


Figure C-1: Schematic Page 1

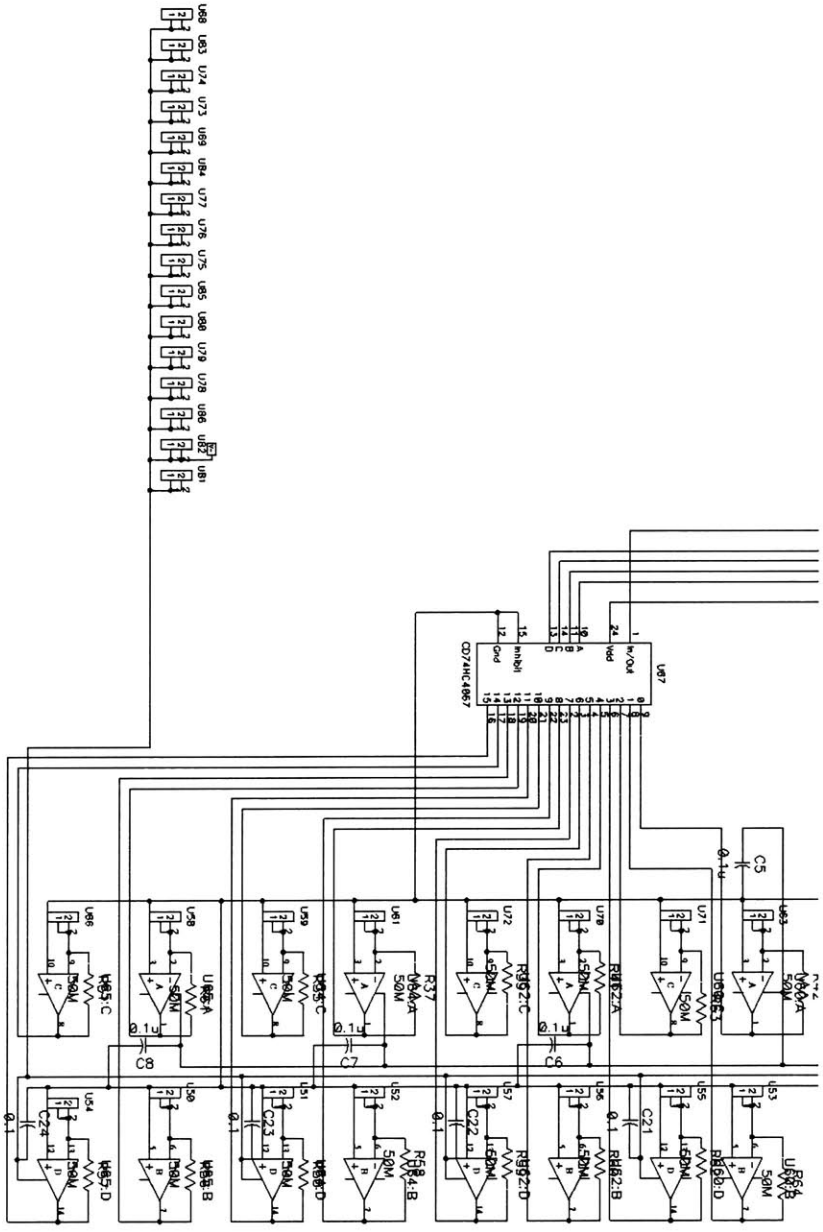


Figure C-2: Schematic Page 2

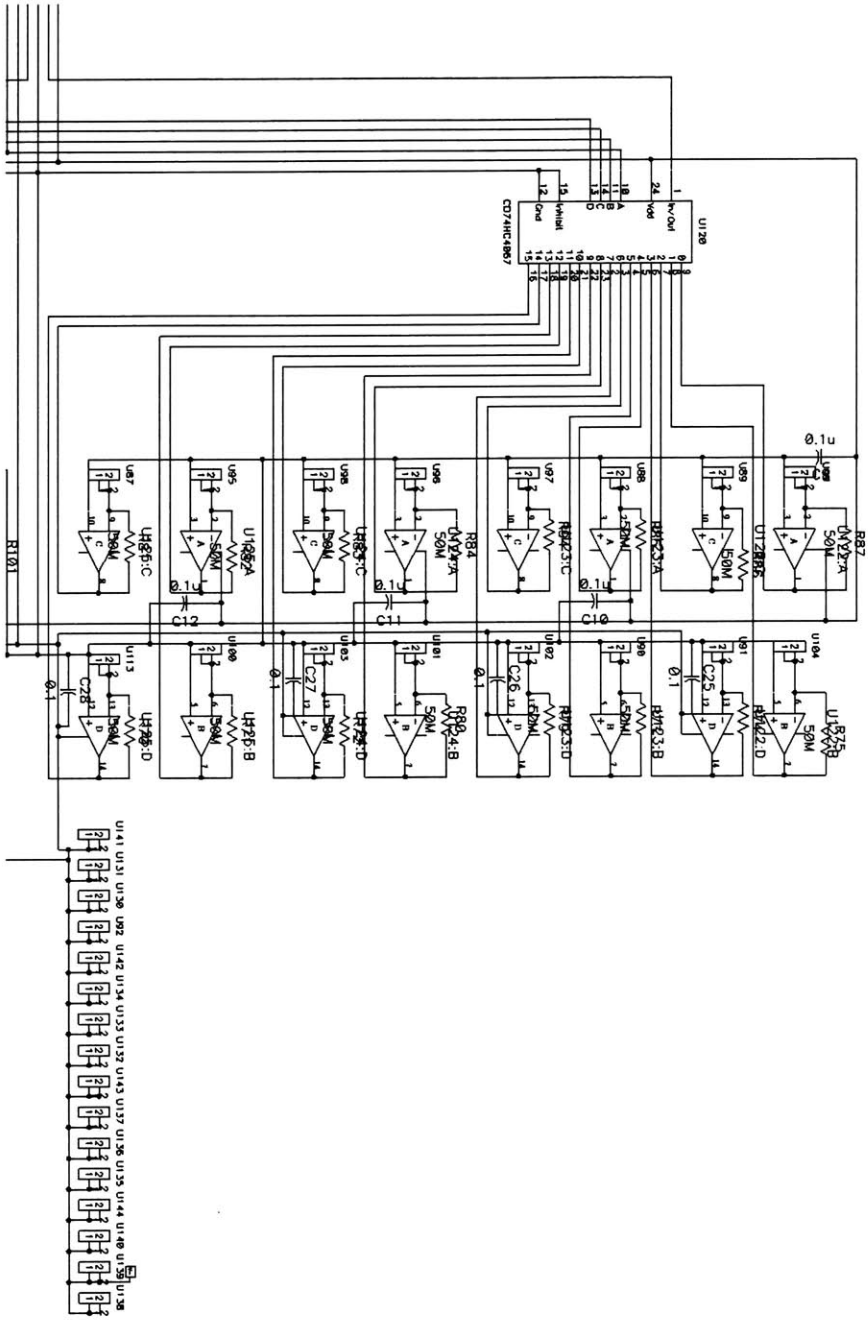


Figure C-3: Schematic Page 3

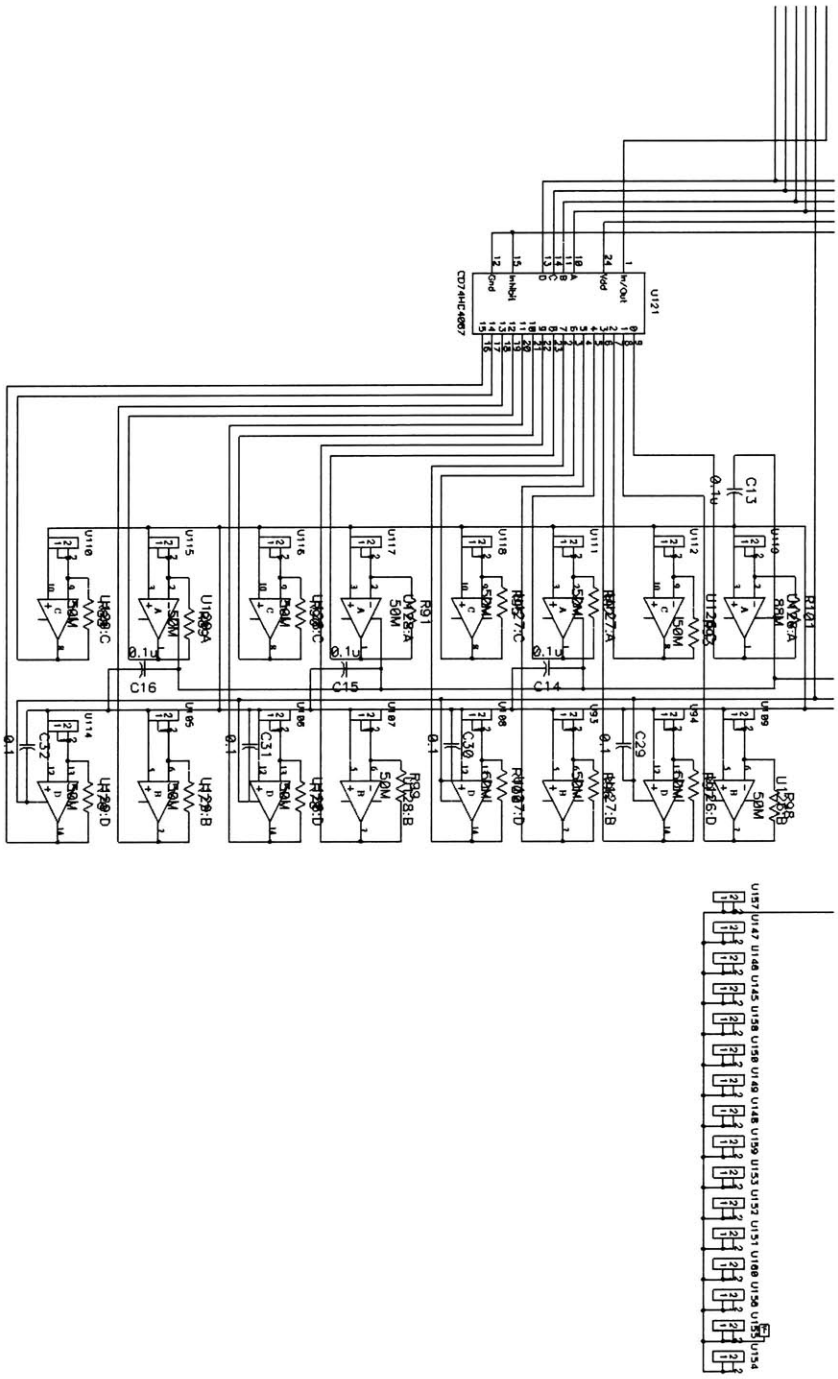


Figure C-4: Schematic Page 4

Bibliography

[1] M. Bayindir, F. Sorin, A. F. Abouraddy, J Viens, S. D. Hart, J. D. Joannopoulos, and Y. Fink. Metal-insultator-semiconductor optoelectronic fibers. *NATURE*, 431:826–829, Oct 14 2004.

[2] H. O. Jacobsm, A. R. Tao, A. Schwartz, Gracias D. H., and G. M. Whitesides. Fabrication of a cylindrical display by patterned assembly. *SCIENCE*, 296:323–325, 2002.

[3] W. Kester, S. Wurcer, and C. Kitchin. *High Impedance Sensors.* Analog Devices, Inc. (www.analog.com).

[4] Compaq, Intel, Microsoft, and NEC. *Universal Serial Bus Specification, Revision 1.1*, Sept. 23 1998. (www.usb.org).

[5] Compaq, Hewlett Packard, Intel, Lucent, Microsoft, NEC, and Phillips. *Universal Serial Bus Specification, Revision 2.0*, Apr. 27 2000. (www.usb.org).

[6] National Semiconductor, Corp. *LMX9820A Bluetooth Serial Port Module*, April 2005. (www.national.com).

[7] A.C. Kak and M. Slaney. *"Prinicples of Computerized Tomographic Imaging"*. IEEE Press, New York, 1988.